

**UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR**



Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

**Falsos positivos en recomendación con bandidos
multi-brazo**

Autor: Emilio Cuesta Fernández

Tutor: Pablo Castells Azpilicueta

enero 2021

Todos los derechos reservados.

Queda prohibida, salvo excepción prevista en la Ley, cualquier forma de reproducción, distribución comunicación pública y transformación de esta obra sin contar con la autorización de los titulares de la propiedad intelectual.

La infracción de los derechos mencionados puede ser constitutiva de delito contra la propiedad intelectual (*arts. 270 y sgts. del Código Penal*).

DERECHOS RESERVADOS

© 19 de enero de 2021 por UNIVERSIDAD AUTÓNOMA DE MADRID
Francisco Tomás y Valiente, n.º 1
Madrid, 28049
Spain

Emilio Cuesta Fernández

Falsos positivos en recomendación con bandidos multi-brazo

Emilio Cuesta Fernández

C\ Francisco Tomás y Valiente N.º 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

AGRADECIMIENTOS

Me ha costado mucho sacar este proyecto adelante y no lo habría conseguido yo solo.

Gracias

a Elisa, por su respaldo;

a Natalia, por su comprensión;

a Pablo y a Javi, por su paciencia y ayuda;

a mis padres, que se han frustrado casi tanto como yo;

y a mis compañeras y compañeros de clase, por cinco años y medio de complicidad.

RESUMEN

Los sistemas de recomendación han ganado mucha importancia en los últimos años con la aparición y el desarrollo de plataformas que ofertan productos a escala masiva. Las personas valoran el hecho de recibir un servicio personalizado que se adecue a sus gustos y necesidades, lo que repercute positivamente tanto en la captación como en la preservación de usuarios. Sin embargo, saber qué recomendar en cada momento es una tarea muy compleja que depende mucho del contexto, de cada uno de los individuos involucrados y del conocimiento que el sistema consigue adquirir sobre ellos a lo largo del tiempo.

Tradicionalmente la evolución de los sistemas de recomendación ha perseguido maximizar la satisfacción de los usuarios, pero recientemente se han identificado contextos donde es más importante evitar a toda costa las recomendaciones desagradables. Aunque pueda parecer contraintuitivo, no se trata de objetivos equivalentes, y, por eso, la consideración de las métricas de falsos positivos ha ganado importancia.

En este proyecto se realiza un estudio experimental para comprender las relaciones presentes entre la evaluación *offline* de sistemas de recomendación empleando las métricas tradicionales, que miden el grado acierto, y la evaluación empleando métricas de falsos positivos, que miden la tasa de error. Se estudia esta perspectiva considerando al mismo tiempo la recomendación como una acción cíclica en el tiempo, donde el sistema recomienda y aprende a la vez. Esta formulación es distinta de la tradicional, que tan solo analiza rondas de recomendación aisladas. La combinación del estudio de falsos positivos con el paradigma cíclico supone un enfoque novedoso que apenas ha sido estudiado hasta el momento. Además, las pruebas realizadas incluyen algoritmos de recomendación basados en bandidos multi-brazo que están a la orden del día. Los bandidos tienen una habilidad natural para balancear la explotación del conocimiento adquirido, lo que permite hacer recomendaciones precisas, y la exploración de nuevas opciones, que permite obtener nueva información.

Los resultados de la fase de experimentación identifican la distribución de la popularidad de los datos de evaluación como un factor clave que puede hacer que los dos tipos de métricas estén de acuerdo o que discrepen radicalmente.

PALABRAS CLAVE

Sistemas de recomendación, sesgos, popularidad, bandidos multi-brazo, falsos positivos

ABSTRACT

In the last few years, recommender systems have gained a lot of importance with the emergence and development of platforms that offer huge amounts of products. Customers appreciate the presence of personalised services that are able to suit their tastes and needs, which have a positive effect for platforms on both attracting and retaining users. However, knowing what to recommend is a complex task that depends on each one of the individuals involved in the purchase process and on the knowledge that the system acquires about them over time.

Traditionally, the evolution of recommender systems has sought to maximise user satisfaction, but some contexts on which it is crucial to avoid unpleasant recommendations have been recently identified. Although it may seem counter-intuitive, these two are not equivalent targets, and consequently the consideration of false-positive metrics has gained importance.

This project contains an experimental study aimed to understand the relationships between the use of traditional metrics, that measure success, and false positive metrics, that measure error rates, as the main criterion in the offline evaluation of a recommender system. This is studied while considering the recommendation as a cyclic action where the system recommends and learns at the same time, which differs from the traditional formulation, that only analyses isolated recommendation rounds. The combination of the study of false positives along with the cyclic paradigm is a novel approach. In addition, the tests carried out include recent recommendation algorithms based on multi-armed bandits. Bandits have a natural ability to balance the exploitation of acquired knowledge, allowing to perform accurate recommendations, and the exploration of new options, allowing more information to be stored.

The obtained results identify the popularity distribution in the test set as a key factor that can cause the two types of metrics to agree or to absolutely disagree.

KEYWORDS

Recommender systems, biases, popularity, multi-armed bandits, false positives

ÍNDICE

1	Introducción	1
1.1	Motivación	2
1.2	Objetivos	2
1.3	Metodología	3
1.4	Estructura del documento	3
2	Estado del arte	5
2.1	Sistemas de recomendación	5
2.1.1	Elementos de un sistema de recomendación	6
2.1.2	Formalización del problema de la recomendación	7
2.1.3	Principales retos de los sistemas de recomendación	8
2.1.4	Taxonomía de los sistemas de recomendación	8
2.1.5	Algoritmos de referencia	10
2.1.6	Evaluación de sistemas de recomendación	10
2.1.7	El sesgo de la popularidad	14
2.2	Bandidos multibrazo	15
2.2.1	Formalización del problema del bandido	16
2.2.2	Estimaciones	17
2.2.3	Estrategias populares	17
2.2.4	Adaptación a sistemas de recomendación	19
3	Trabajo previo	21
3.1	Una plataforma de recomendación: CycloRec	21
3.2	Análisis de los conjuntos de datos	23
3.2.1	MovieLens100k	23
3.2.2	CM100k	24
4	Experimentación	27
4.1	Metodología general	27
4.2	Experimento 1	28
4.3	Experimento 2	32
4.3.1	Familiar CM100k	32
4.3.2	CM100k	34
5	Conclusiones y limitaciones	37

5.1 Conclusiones	37
5.2 Limitaciones y trabajo futuro	38
Bibliografía	40
Acrónimos	41
Apéndices	43
A Pseudocódigos	45
A.1 Item Bandit	45
A.2 Ignore Missing Item Bandit	46
A.3 kNN Bandit	46

LISTAS

Lista de algoritmos

A.1	Item Bandit	45
A.2	IM Item-Bandit update	46
A.3	kNN Bandit	46

Lista de ecuaciones

2.1	Función de utilidad	7
2.2	Elección natural de un ítem	7
2.3	Recomendación Avg	10
2.4	Precision	13
2.5	Recall	13
2.6	AntiPrecision	13
2.7	Fallout	13
2.8	Precision vs Antiprecision	13
2.9	Recall vs Fallout	13
2.10	Coeficiente de correlación de Kendall	14
2.11	Estimación del valor de cada brazo	17
2.12	ϵ -greedy	17
2.13	Upper Confidence Bound	18
2.14	Thompson Sampling	19
2.15	1NN-Bandit: Elección del recomendador	20
2.16	1NN-Bandit: Elección del ítem	20
2.17	kNN-Bandit: Elección del ítem	20

Lista de figuras

2.1	Sistema de recomendación	5
2.2	Matriz de utilidad	6
2.3	Tipos de algoritmos de recomendación	9
2.4	<i>Ratings</i> en MovieLens100k	15

2.5	El dilema del bandido	16
2.6	Funciones de densidad de algunas distribuciones Beta	19
3.1	Bloques de <i>CycloRec</i>	22
3.2	<i>Ratings</i> en MovieLens100k	24
3.3	<i>Ratings</i> en CM100k	25
4.1	Recall normalizado en ML100k	30
4.2	Fallout normalizado en ML100k	30
4.3	Rankings de algoritmos en ML100k	31
4.4	Recall normalizado en FAMCM100k	33
4.5	Fallout normalizado en FAMCM100k	34
4.6	Rankings de algoritmos en FAMCM100k	34
4.7	Recall normalizado en CM100k	35
4.8	Fallout normalizado en CM100k	36
4.9	Rankings de algoritmos en CM100k	36

Lista de tablas

3.1	Algoritmos incluidos en la librería <i>CycloRec</i>	23
3.2	Descripción de MovieLens100k y CM100k	23
4.1	Rejilla para cada parámetro	28
4.2	Configuraciones que maximizan Recall en $t = 500$ para MovieLens100k	29
4.3	Configuraciones que minimizan Fallout en $t = 500$ para MovieLens100k	29
4.4	Configuraciones que maximizan Recall en $t = 500$ para Familiar CM100k	32
4.5	Configuraciones que minimizan Fallout en $t = 500$ para Familiar CM100k	33
4.6	Configuraciones que maximizan <i>Recall</i> y minimizan <i>Fallout</i> en $t = 500$ para CM100k .	35

INTRODUCCIÓN

Los sistemas de recomendación (RS) están en auge. Lo están porque los servicios personalizados son, en general, atractivos para los usuarios, que quieren ahorrar tiempo en sus búsquedas; para los proveedores, que pueden aumentar la difusión de sus productos; y para los intermediarios, porque les facilitan obtener y retener más usuarios y proveedores.

Los RS ofrecen una serie de “ítems”, que normalmente pertenecen a un catálogo de gran volumen, a los usuarios del sistema con el objetivo de satisfacerles. Es el caso de las recomendaciones de *Amazon* (productos variados), *Netflix* (películas y series), *Spotify* (canciones y podcasts) o *Google Ads* (anuncios), por citar algunos entre cientos de ejemplos. Sin embargo, la elección de los ítems adecuados para cada uno de los usuarios no es trivial. Existen muchos algoritmos para seleccionarlos que tienen enfoques muy diversos: basados en la popularidad, en las características de los usuarios, con más o menos componentes aleatorios etc. Además, están surgiendo continuamente ideas que atienden a otros factores o utilizan técnicas novedosas.

Sin embargo, casi todos estos planteamientos tienen algo en común: se centran únicamente en maximizar la satisfacción del usuario. Las métricas que miden el éxito de un sistema de recomendación se basan en el número de aciertos y suelen ignorar los fallos, aun sabiendo que el coste de una mala recomendación puede ser desastroso en determinados ámbitos. Además, aunque intuitivamente los resultados que se obtienen al maximizar la satisfacción deberían ser coherentes con los que se obtienen al minimizar la insatisfacción, esto no siempre es así. ¿Qué circunstancias provocan esta discrepancia? ¿Cuál de las dos opciones es mejor?

En este proyecto se estudia el origen de estas diferencias en base a los resultados de varios algoritmos de recomendación. Tienen un peso especial los planteamientos basados en bandidos multi-brazo (MAB), que en los últimos años han ganado mucha relevancia por su capacidad natural para afrontar algunos problemas clave en el mundo de los sistemas de recomendación.

1.1. Motivación

El principal motivo para llevar a cabo este proyecto es la escasez de estudios que tratan de minimizar las recomendaciones desagradables para los usuarios. Una gran mayoría de la literatura se centra en maximizar la satisfacción e ignora los efectos que puede tener una mala recomendación y, aunque es razonable esperar cierta correlación, algunas publicaciones observan discrepancias cuando se tienen en cuenta los dos criterios [Cañamares and Castells, 2018; Mena-Maldonado et al., 2020]. Por si fuera poco, existen varios escritos en el ámbito de la psicología que afirman que una mala recomendación puede tener un impacto negativo en el grado de implicación y satisfacción de un usuario con respecto a una aplicación, incluso mayor que el refuerzo positivo que supondrían varias recomendaciones acertadas [Baumeister et al., 2001; Yin et al., 2010]. Esto hace que sea muy relevante empezar a tener en cuenta el número de errores que cometen los sistemas de recomendación y sugiere una revisión de algunos enfoques experimentales tradicionales. Además, teniendo en cuenta la gran variedad de contextos en los que se están empleando los RS actualmente, es conveniente saber adaptarlos para satisfacer objetivos de diversa índole.

Por otro lado, en este trabajo se contempla la perspectiva de la recomendación como un problema del tipo MAB porque, recientemente, han sido muchas las publicaciones que han propuesto algoritmos de recomendación basados en *bandits*, obteniendo además buenos resultados con enfoques relativamente simples [Cañamares et al., 2019; Dragone et al., 2019; Kawale et al., 2015; Li et al., 2010; Li et al., 2016; McInerney et al., 2018; Sanz-Cruzado et al., 2019; Wang et al., 2019; Zhao et al., 2013]. Es reseñable que estos algoritmos son capaces de obtener nueva información al mismo tiempo que sacan partido de la información ya conocida, y esta dualidad es esencial en los sistemas de recomendación. Su actual importancia en las tendencias innovadoras es el principal motivo para trabajar con ellos.

1.2. Objetivos

El objetivo principal de este proyecto es identificar cuándo discrepan las métricas que miden la satisfacción de los usuarios con aquellas que miden su descontento. Para lograrlo, también será necesario abordar los siguientes:

- Estudiar las relaciones entre las métricas tradicionales, que miden la satisfacción del usuario, y las métricas basadas en falsos positivos, que se pueden emplear para medir las recomendaciones insatisfactorias.
- Diseñar e implementar una librería que permita ejecutar procesos de recomendación iterativos sobre distintos conjuntos de datos. Esta librería se documentará en detalle y se hará pública en *GitHub* para que cualquier persona pueda emplearla. Se incluirán, en un primer momento, algunos de los algoritmos de recomendación más sencillos para poder emplear-

los como referencia. Estos son: recomendación aleatoria, por popularidad y por puntuación media.

- Estudio en profundidad del problema del bandido multi-brazo, incluyendo tres de las estrategias más conocidas para afrontarlo: ϵ -greedy, *upper confidence bound* (UCB) y *Thompson sampling* (TS) .
- Entender e implementar en la librería dos algoritmos de recomendación basados en *bandits*, uno personalizado y otro no personalizado.
- Estudiar las distribuciones de popularidad más comunes en conjuntos de datos de recomendación para observar el comportamiento de las métricas, tanto de verdaderos como de falsos positivos, con cada una de ellas.

1.3. Metodología

Como se puede observar en la sección anterior, la mayor parte de los objetivos tiene un enfoque doble: en primer lugar es necesario realizar un estudio teórico para, a posteriori, realizar las implementaciones o experimentaciones necesarias. En este trabajo se ha optado por realizar pequeñas iteraciones teórico-prácticas, alternando, en todas ellas, fases de investigación con fases de codificación y experimentación.

La primera iteración se corresponde con lecturas introductorias a los RS y la implementación de los algoritmos básicos de referencia. Al tratarse de la primera fase de codificación, también se incluye aquí el diseño de una librería que sea fácilmente extensible con otros algoritmos y aplicable a cualquier conjunto de datos. En la segunda iteración se han estudiado los algoritmos MAB más conocidos y dos formas diferentes de aplicarlos para lograr buenas recomendaciones. También se ha ampliado la librería anterior con uno de los mismos. En la tercera etapa, se ha revisado la documentación relacionada con el descontento de los usuarios, se han propuesto experimentos y se han ejecutado algunos de los algoritmos disponibles sobre conjuntos de datos con propiedades distintas, para ver bajo qué condiciones las métricas de satisfacción y de desagrado son coherentes entre sí.

1.4. Estructura del documento

Este documento consta de las siguientes secciones:

- 1.– **Introducción.** Enumera brevemente los principales motivos para la elaboración de este proyecto así como los objetivos que se espera satisfacer. También contiene un breve resumen de la metodología empleada durante el desarrollo del trabajo.
- 2.– **Estado del arte.** Contiene los conceptos teóricos esenciales para la comprensión de

este proyecto. Consta de una sección introductoria a los sistemas de recomendación, prestando especial atención a las distintas métricas que se pueden emplear para evaluar la calidad de las recomendaciones. También describe el problema del bandido multi-brazo y cómo se pueden aplicar a los RS .

3.— **Trabajo previo.** Introduce el funcionamiento de la librería que se ha desarrollado para realizar los experimentos. También contiene un análisis de las propiedades de los conjuntos de datos que se han empleado.

4.— **Experimentación y resultados.** Describe los experimentos que se han realizado y analiza sus resultados.

5.— **Conclusiones.** En esta sección se resumen los resultados más relevantes del proyecto y se esbozan líneas cuya investigación pueda resultar interesante en el futuro.

ESTADO DEL ARTE

2.1. Sistemas de recomendación

Hoy en día, las redes sociales, los servicios de *streaming* o las plataformas de *e-commerce* ofrecen cantidades ingentes de productos e información. Para evitar tener que realizar búsquedas exhaustivas, algunos usuarios agradecen que se les muestre un subconjunto de la oferta de acuerdo a sus intereses y necesidades puntuales. Incluso cuando el volumen de datos no es tan elevado, la presencia de recomendaciones personalizadas sirve para mejorar la experiencia de los usuarios de una aplicación y facilita su uso. Así, para las plataformas de este tipo surge la necesidad de disponer de un agente capaz de obtener sugerencias que interesen a sus clientes. Estos agentes son lo que conocemos como Sistemas de Recomendación (RS).

Agradar a un usuario es una tarea compleja, sobre todo porque se trata de un concepto que depende tanto del individuo como del ámbito de la aplicación. Por ello, los RS necesitan recopilar información acerca del comportamiento y de las reacciones de cada uno de los usuarios, tanto a las recomendaciones ofrecidas con anterioridad como de los productos que puedan encontrar y consumir por su cuenta. A partir de esa información, los sistemas de recomendación han de ser capaces de ir mejorando su precisión paulatinamente. Este proceso se ilustra en la Figura 2.1 para dos usuarios.

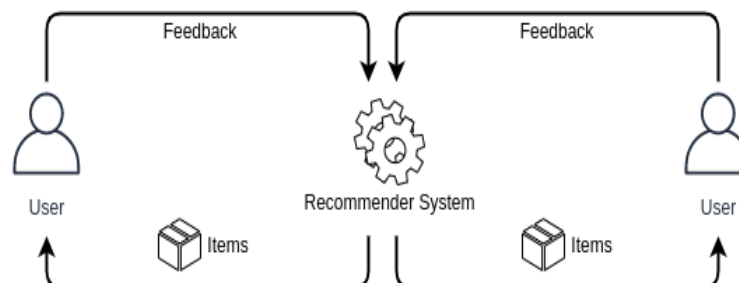


Figura 2.1: Comportamiento de un sistema de recomendación de 2 usuarios

2.1.1. Elementos de un sistema de recomendación

En cualquier sistema de recomendación se pueden encontrar los siguientes elementos:

- \mathcal{U} es el conjunto de usuarios u del sistema, aquellos que van a recibir las recomendaciones.
- \mathcal{I} es el conjunto de ítems o productos i del sistema, que serán recomendados a los usuarios de \mathcal{U} .
- \mathcal{R} es el conjunto de *ratings* conocidos. Un *rating* es esencialmente la valoración que un usuario otorga a un ítem. \mathcal{R} contiene tuplas (u, i, r_{ui}) en la que se almacena el *feedback* r_{ui} dado por el usuario u al ítem i .

Es común que los usuarios valoren los productos de forma explícita, asignando una puntuación en una escala predeterminada, pero no siempre es así. Por ejemplo, en servicios de *streaming* una de las medidas de agrado más importantes es el tiempo de reproducción de un vídeo y eso es algo que el sistema almacena de forma automática. Se distingue entonces entre **feedback explícito**, cuando el usuario indica manualmente lo mucho o poco que le ha gustado un producto, y **feedback implícito**, cuando este valor se deduce automáticamente a partir de su comportamiento.

Toda esta información se puede organizar en una **matriz de utilidad** como la de la Figura. 2.2









	Item 1	Item 2	Item 3	...	Item M
				...	
User 1 	3	5	?	...	?
User 2 	1	?	0	...	4
User 3 	5	?	4	...	?
...
User N 	?	?	2	...	2

Figura 2.2: Matriz de utilidad

Como, normalmente, el número de ítems es muy elevado y un usuario consume solo una pequeña parte de los mismos, las matrices de utilidad tienden a ser muy dispersas, siendo desconocidos la mayor parte de los *ratings*. Es importante tener cuidado con los valores que se desconocen o *missing values*, porque es muy distinto no disponer de información que asumir que existe una puntuación, aunque sea un promedio. El gran tamaño de las matrices de utilidad hace que muchas operaciones con ellas sean muy costosas y no sea viable realizar los cálculos cada vez que se obtiene nueva información, por eso hay que tener cuidado con la eficiencia de los RS y su capacidad para mantenerse actualizados.

2.1.2. Formalización del problema de la recomendación

Una de las formalizaciones clásicas del problema de la recomendación [Adomavicius and Tuzhilin, 2005] se basa en el concepto de **función de utilidad**, una función g tal que:

$$\begin{aligned} g : \mathcal{U} \times \mathcal{I} &\rightarrow \hat{R} \\ g(u, i) &\mapsto \hat{r}_{ui} \end{aligned} \tag{2.1}$$

donde \hat{R} es un conjunto totalmente ordenado y \mathcal{U} e \mathcal{I} los conjuntos de usuarios y de ítems ya mencionados. De forma intuitiva, g asocia a cada tupla (u, i) un valor que indica cómo de útil sería el ítem i para el usuario u y, a la hora de recomendar a cada usuario $u \in \mathcal{U}$, bastará con elegir el ítem $j \in \mathcal{I}$ que cumpla:

$$j = \operatorname{argmax}_{i \in \mathcal{I}} g(u, i) \tag{2.2}$$

Por tanto, una vez elegida una función de utilidad, el proceso de recomendación es muy sencillo. El problema radica en definir una g aplicable a todas las tuplas $(u, i) \in \mathcal{U} \times \mathcal{I}$ y que ofrezca buenos resultados. Normalmente, se define una aplicación que sea coherente con los *ratings* almacenados en \mathcal{R} (es común tomar como valores de utilidad los propios ratings, i.e. $g(u, i) = \hat{r}_{ui} = r_{ui}$, pero existen muchas otras opciones válidas), y posteriormente se extiende al resto de combinaciones de $\mathcal{U} \times \mathcal{I}$ para estimar las utilidades que no conocemos. En realidad, fijado un usuario u , este problema es equivalente a un problema de *ranking* del área de la *Information Retrieval*.

En muchas ocasiones será necesario realizar pequeñas modificaciones o imponer condiciones adicionales sobre el problema. Por ejemplo, es frecuente evitar que un mismo usuario reciba la misma recomendación más de una vez, sobre todo en enfoques más académicos. También puede variar el número de ítems a recomendar.

Generalización a un proceso cíclico

Como se ilustra en la Figura 2.1, el proceso de recomendación también se puede plantear como un problema cíclico. De esta manera, por cada recomendación realizada se recibe un *feedback* que se almacena en \mathcal{R} y se emplea para realizar mejores recomendaciones.

Con este enfoque, la formalización del problema cambia ligeramente. En un proceso de N ciclos, en cada iteración se puede emplear una función de utilidad distinta $(g^1, g^2, g^3, \dots, g^N)$, lo que permite adaptar los valores de utilidad de acuerdo a la nueva información recibida. A efectos prácticos, es suficiente con definir un método para que los valores \hat{r}_{ui} se actualicen en función de la valoración recibida en cada etapa.

2.1.3. Principales retos de los sistemas de recomendación

El problema del arranque en frío

Una de las grandes limitaciones de los sistemas de recomendación es que, cuando no hay apenas datos, es muy difícil realizar recomendaciones acertadas. Esto se debe, entre otros motivos, a que los datos recopilados no son suficientes para reflejar los gustos de los usuarios, o a que las estimaciones que se pueden hacer sobre los mismos son muy sesgadas. Esta situación es lo que se conoce como **problema del arranque en frío**. Para solucionar este problema, es importante que los algoritmos de recomendación sean capaces de realizar ofrecimientos a priori arriesgados con el fin de recopilar nueva información. Así, a largo plazo, se aumenta el conocimiento del sistema y la certeza sobre el mismo.

Exploración y explotación

El hecho de tener que arriesgarse para obtener información da lugar a otro de los problemas más importantes a los que se enfrentan los RS, que es establecer un balance adecuado entre **exploración y explotación**. Se está explorando cuando las recomendaciones que se ofrecen a un usuario se realizan con el objetivo de ampliar o ratificar el conocimiento sobre sus gustos, y así poder ofrecer mejores opciones a largo plazo. Se está explotando cuando un RS emplea la información de la que dispone con el fin de maximizar exclusivamente la probabilidad de agrandar al usuario con la siguiente recomendación. Si se abusa de la exploración, que tiene una alta probabilidad de fallo, es posible que los usuarios de un sistema se cansen de recibir malas recomendaciones y dejen de utilizarlo. Por otro lado, si se abusa de la explotación los resultados serán monótonos y poco diversos, lo que también puede desagradar o provocar indiferencia en el usuario, es más, podrían no encontrarse nunca las opciones que más le gustan, quedando el sistema estancado en pequeños máximos locales. Por eso es importante ajustar las proporciones de exploración y explotación de acuerdo a la información disponible en cada momento.

2.1.4. Taxonomía de los sistemas de recomendación

En primer lugar, es posible clasificar los RS como personalizados o no personalizados. Si el resultado del algoritmo de recomendación (el ítem recomendado) no depende del usuario objetivo, el sistema será no personalizado; en el caso contrario nos encontraremos ante una recomendación personalizada. Formalmente, si g es la función de utilidad del sistema y $\forall i \in \mathcal{I} \forall u, v \in \mathcal{U} g(u, i) = g(v, i)$, entonces las recomendaciones serán no personalizadas. Normalmente los algoritmos personalizados ofrecen mejores resultados, al adecuarse más a las necesidades específicas de un usuario. Sin embargo, en situaciones como la de un arranque en frío y la falta de información que ello conlleva, es posible que un algoritmo no personalizado sea más conveniente.

El resto de la clasificación se describe en la Figura 2.3. En ella, los nodos intermedios constituyen categorías que se describen a continuación y las hojas son ejemplos de algunos algoritmos destacados.

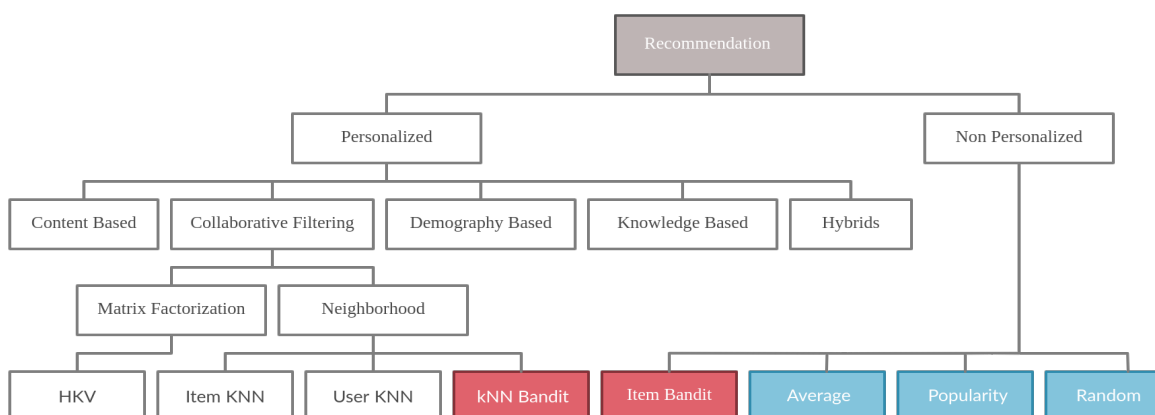


Figura 2.3: Tipos de algoritmos de recomendación

Las subcategorías de los algoritmos personalizados se definen de acuerdo a lo propuesto en [Burke, 2007] y [Ricci et al., 2011]. En esa clasificación se encuentran los siguientes grupos:

- **Content based:** El sistema recomienda a cada usuario ítems basados en los ítems que le han gustado en el pasado. La similitud de los ítems se computa basándose en sus atributos. Por ejemplo, en un contexto de librerías, un atributo podría ser el género y otro el autor de un libro.
- **Collaborative filtering:** Los filtros colaborativos se basan en la creencia en que, si un usuario coincide con otro en muchas de las valoraciones que han ambos han dado en el pasado, es probable que coincidan también en las valoraciones que ha realizado solo uno de los dos. De esta manera, se puede obtener un catálogo de ítems potencialmente exitosos para la mayoría de usuarios. Dentro de este grupo encontraríamos otras dos categorías muy extendidas:
 - **Matrix factorization:** Esta estrategia se basa principalmente en el cálculo de la descomposición por valores singulares de la matriz de utilidad. A partir de ella, será posible reconstruir una matriz más completa que contiene estimaciones de *ratings* para cada tupla. Estas estrategias son, por regla general, bastante costosas a nivel computacional. El algoritmo HKV propuesto en [Hu et al., 2008] es uno de los más extendidos y utilizados en la literatura. En [Pilászy et al., 2010] se propone una versión similar pero más eficiente.
 - **Neighborhood models:** Son modelos basados en vecindarios, como *k nearest neighbors* (KNN) . Se pueden construir vecindarios tanto de usuarios como de ítems, para ello, se emplean directamente los *ratings* presentes en la matriz de

utilidad. Para calcular las distancias es común emplear la similitud coseno. Un enfoque novedoso viene dado en [Sanz-Cruzado et al., 2019], en el que se utilizan vecindarios en combinación con bandidos multi-brazo (Sección 2.2.4).

- **Demography based:** Estos algoritmos asumen que las personas con rasgos demográficos comunes tienen gustos similares. Estos rasgos pueden ser la geolocalización habitual de los individuos, su edad o su sexo, entre muchos otros.
- **Knowledge based:** A través del conocimiento de las necesidades puntuales de los usuarios, que pueden ser introducidas de forma externa o estimadas por el sistema, se buscan los ítems más adecuados. Es un enfoque distinto en el que ya no importa tanto el usuario en sí como las necesidades que tenga en un momento determinado.
- **Hybrid:** Cualquier algoritmo que combine dos o más de las estrategias anteriores.

2.1.5. Algoritmos de referencia

Es común tomar como referencia algunos algoritmos muy simples tanto para detectar anomalías (e.g. peor efectividad que una recomendación aleatoria), como para apreciar el grado objetivo de mejora que un algoritmo está realmente consiguiendo (e.g. por encima de opciones triviales como contar el número de *ratings*). Esto permite ver si sistemas de recomendación más complejos merecen la pena tanto en términos de acierto como en términos de coste computacional. Las tres referencias o *baselines* más comunes son:

- **Random (Rnd):** En cada ronda se recomienda un ítem al azar.
- **Popularity (Pop):** En cada ronda se recomienda el ítem que más *ratings* positivos haya recibido hasta el momento.
- **Average (Avg):** En cada ronda se recomienda el ítem que mejor *rating* promedio tenga. Los promedios σ_i se calculan realizando un suavizado aditivo tal y como se indica en la Ecuación 2.3, donde a_i es la suma de las n_i valoraciones del ítem i :

$$\sigma_i = \frac{a_i + 1}{n_i + 2} \quad (2.3)$$

2.1.6. Evaluación de sistemas de recomendación

Evaluar un sistema de recomendación no es tan sencillo como limitarse a valorar el nivel de acierto de las recomendaciones que realiza. La precisión es necesaria, pero no suficiente. Existen muchos otros factores que pueden ser muy importantes para los usuarios: el tiempo de respuesta, la diversidad de las propias recomendaciones, la capacidad del RS para adaptarse a nuevas modas, o, incluso, hacer ver que no se está invadiendo su privacidad. Por tanto, en función del ámbito de aplicación de

cada RS será necesario emplear unos criterios u otros, cuya identificación no siempre es inmediata [Ricci et al., 2011].

Más allá de los criterios que se empleen, existen dos principales métodos, complementarios entre sí, para valorar el rendimiento de un sistema de recomendación: la **evaluación online** y la **evaluación offline**.

La evaluación *online* simula una situación real, típicamente con el sistema en un entorno de producción. De esta forma, los usuarios interactúan con el sistema, valorando las recomendaciones recibidas, una vez que el experimento ha comenzado. Existen varias variantes, por ejemplo emplear *A/B testing*, pero el punto fuerte de todas ellas es que se basan en el criterio de los usuarios en un entorno real. Sin embargo, juntar un número significativo de personas es costoso y obtener resultados relevantes puede tomar bastante tiempo.

La evaluación *offline* se realiza utilizando *ratings* recopilados con anterioridad. Así, toda participación por parte de los usuarios ocurre antes de que de comienzo el experimento. Con esos datos, es posible simular un proceso de recomendación-*feedback* y así evaluar tanto la calidad de las recomendaciones realizadas por un algoritmo como adquirir información acerca del comportamiento de un usuario, bajo la asunción de que dicho comportamiento será parecido al que presente un entorno interactivo real. En [Cañamares et al., 2020] se presenta una recopilación muy detallada de consideraciones a la hora de realizar una evaluación *offline*. En general, este proceso ofrece resultados menos fidedignos que una evaluación *online*, pero su fácil automatización y su bajo coste hacen de la evaluación *offline* el método más extendido para realizar validaciones y comparaciones de algoritmos, en especial en el mundo académico.

Es común realizar una primera fase de evaluación *offline* que permita seleccionar los RS con más potencial, para posteriormente volver a evaluarlos en un entorno real (*online*) y seleccionar el que mejores resultados ofrezca.

Una práctica común es evaluar un sistema de recomendación dividiendo el conjunto de *ratings* en tres subconjuntos disjuntos: uno para entrenar, otro para validar y otro para el test final. En [Bellogín et al., 2017] se estudian diversas formas para realizar estas particiones y el impacto que pueden tener en los resultados obtenidos. También existe la opción de sustituir el proceso final por una evaluación *online*.

Evaluando fallos

En cualquier caso, es esencial hablar de las métricas de evaluación. De entre todos los factores que determinan el éxito de una recomendación, el “acierto” ha sido tradicionalmente el más relevante. Dos de las métricas de verdaderos positivos (métricas TP) más extendidas para medirlo son *Precision* y *Recall*. Sin embargo, en los últimos años las métricas que miden los fallos evidentes, conocidas co-

mo métricas FP, han captado la atención de importantes sectores del negocio. Esto se debe a que en determinadas circunstancias, una mala recomendación puede tener un impacto muy negativo en la experiencia de un usuario. Por ejemplo, estudios recientes de empresas líderes en recomendación, como *Spotify*, demuestran que la mayoría de los usuarios cambian de *playlist* al encontrarse con canciones que les desagradan, algo que no ocurre con canciones que les son indiferentes [Brost et al., 2019] [Fields, 2011]. Los falsos positivos también causan un rechazo importante en usuarios de servicios de citas online [Pizzato et al., 2013] o en servicios de anuncios online [Bron et al., 2019]. Así, cobran importancia métricas como *antiPrecision* y *Fallout*, que son, respectivamente, las antimétricas de *Precision* y *Recall*. Una antimétrica es la adaptación natural que se hace de una métrica de verdaderos positivos para comenzar a medir falsos positivos [Sánchez and Bellogín, 2018].

Métricas y relaciones

Para entender cómo se calculan estas métricas se introducen a continuación unas nociones básicas. Sea \mathcal{T} el conjunto de evaluación, es decir, un conjunto que contiene *ratings* tanto positivos, que hay que descubrir, como negativos, que se deberían evitar; y sea λ el **umbral de relevancia**, i.e. el valor a partir del cuál se considera que un *rating* es positivo y por debajo del cual se considera negativo.

En un proceso de un paso, tan solo hay que calcular la proporción de recomendaciones con un *rating* positivo o negativo en \mathcal{T} para obtener Precision o AntiPrecision, respectivamente. De igual manera, basta con calcular el grado de descubrimiento instantáneo de *ratings* positivos o negativos de \mathcal{T} para obtener Recall o Fallout.

Sin embargo, en procesos de recomendación cíclicos, es normal trabajar con las versiones acumulativas de las métricas anteriores. Para definir las, se consideran el conjunto \mathcal{R}_t , que es el conjunto que contiene todas las recomendaciones realizadas hasta el instante de tiempo t . Así, se pueden calcular los siguientes valores:

- **True Positives (TP)** : Número de recomendaciones realizadas hasta el instante t y que tienen un feedback positivo asociado en \mathcal{T} . Es decir:

$$TP_t = |\{(u, i) \in \mathcal{R}_t \mid (u, i, r) \in \mathcal{T} \wedge r \geq \lambda\}|$$

- **False Positives (FP)** : Número de recomendaciones realizadas hasta el instante t y que tienen un feedback negativo asociado en \mathcal{T} . Es decir:

$$FP_t = |\{(u, i) \in \mathcal{R}_t \mid (u, i, r) \in \mathcal{T} \wedge r < \lambda\}|$$

- **True Negatives (TN)** : Número de valoraciones negativas por descubrir en el instante t . Es decir:

$$TN_t = |\{(u, i) \notin \mathcal{R}_t \mid (u, i, r) \in \mathcal{T} \wedge r < \lambda\}|$$

- **False Negatives (FN)** : Número de valoraciones positivas por descubrir en el instante t . Es

decir:

$$FN_t = |\{(u, i) \notin \mathcal{R}_t \mid (u, i, r) \in \mathcal{T} \wedge r \geq \lambda\}|$$

En base a ellos, se definen las métricas acumulativas de acierto:

- **Precision:** Proporción de aciertos en las recomendaciones realizadas.

$$Prec(t) = \frac{|\{(u, i) \in \mathcal{R}_t \mid (u, i, r) \in \mathcal{T} \wedge r \geq \lambda\}|}{|\mathcal{R}_t|} = \frac{TP_t}{|\mathcal{R}_t|} \quad (2.4)$$

- **Recall:** Grado de descubrimiento de las valoraciones positivas.

$$Rec(t) = \frac{|\{(u, i) \in \mathcal{R}_t \mid (u, i, r) \in \mathcal{T} \wedge r \geq \lambda\}|}{|\{(u, i, r) \in \mathcal{T} \mid r \geq \lambda\}|} = \frac{TP_t}{TP_t + FN_t} \quad (2.5)$$

Y sus correspondientes antimétricas:

- **AntiPrecision:** Proporción de fallos en las recomendaciones realizadas.

$$APrec(t) = \frac{|\{(u, i) \in \mathcal{R}_t \mid (u, i, r) \in \mathcal{T} \wedge r < \lambda\}|}{|\mathcal{R}_t|} = \frac{FP_t}{|\mathcal{R}_t|} \quad (2.6)$$

- **Fallout:** Grado de descubrimiento de las valoraciones negativas.

$$Fall(t) = \frac{|\{(u, i) \in \mathcal{R}_t \mid (u, i, r) \in \mathcal{T} \wedge r < \lambda\}|}{|\{(u, i, r) \in \mathcal{T} \mid r < \lambda\}|} = \frac{FP_t}{FP_t + TN_t} \quad (2.7)$$

Suponiendo que se está trabajando sobre un conjunto de datos en el que todos los usuarios han valorado todos los ítems (i.e. $\forall u \forall i \exists r (u, i, r) \in \mathcal{T}$), es posible establecer las siguientes relaciones entre las métricas TP y sus respectivas métricas FP:

$$\forall t \quad Prec(t) + APrec(t) = \frac{TP_t}{|\mathcal{R}_t|} + \frac{FP_t}{|\mathcal{R}_t|} = 1 \quad (2.8)$$

$$\forall t \quad (TP_t + FN_t)Rec(t) + (FP_t + TN_t)APrec(t) = TP_t + FP_t = |\mathcal{R}_t| \quad (2.9)$$

Bajo el supuesto anterior, las ecuaciones (2.8) y (2.9) muestran que existe una relación lineal inversa entre sus respectivas métricas. Esto es coherente: a mayor acierto se espera haber fallado menos. Sin embargo, la condición de completitud no es muy útil en el ámbito de los sistemas de recomendación, porque es un caso que no se da nunca y que reduce el problema al absurdo. En una situación real no se pueden verificar esas relaciones. De hecho, [Mena-Maldonado et al., 2020] comprueba que, en un proceso de recomendación de un paso, el desconocimiento de algunos *ratings* no sólo provoca que los valores de *Precision* y *AntiPrecision* no se comporten de acuerdo a (2.8), sino que se llega a invertir la tendencia (más acierto implica más fallo) y aparecen discrepancias sistemáticas entre las dos métricas. Esto concuerda con lo observado en estudios como [Cañamares and Castells, 2018].

Correlación de Kendall

Una vez evaluados distintos sistemas de recomendación es normal ordenarlos según sus resultados. Utilizar dos criterios, como dos métricas distintas, puede dar lugar a diferencias en esas ordenaciones. Para medir el grado de acuerdo o desacuerdo entre dos *rankings*, se suele usar el coeficiente de correlación de *Kendall*. Si se ordenan N elementos y las posiciones de cada elemento i en cada *ranking* vienen dadas por x_i y y_i respectivamente, entonces se define el coeficiente de *Kendall* como:

$$\tau = \frac{\sum_{i=1}^N \sum_{j=1}^{i-1} \text{sgn}(x_i - x_j) \cdot \text{sgn}(y_i - y_j)}{\binom{N}{2}} \quad (2.10)$$

τ toma un valor de 1 cuando las dos ordenaciones son idénticas y de -1 cuando son totalmente inversas, es decir, cuando el primer elemento del primer *ranking* es el último del segundo, el segundo es el penúltimo y así sucesivamente.

En este trabajo utilizaremos la correlación de *Kendall* para ver el grado de desacuerdo entre los *rankings* de sistemas que proporcionan recall y fallout, respectivamente.

2.1.7. El sesgo de la popularidad

La mayoría de los conjuntos de datos que se emplean en el ámbito de la recomendación provienen de situaciones de uso reales y, por tanto, suelen presentar sesgos de popularidad motivados por el comportamiento natural de los usuarios. Así, unos pocos ítems, los más populares, acumulan una proporción mayoritaria de las valoraciones mientras que los demás han recibido poco *feedback*. Trabajar con estos sesgos es razonable porque representan un contexto auténtico, pero no por ello deja de ser relevante estudiar y comprender sus efectos. Al emplear datos sesgados, los algoritmos de recomendación tienden a reproducir el sesgo y así favorecer a los ítems más populares. En muchas ocasiones aprovecharse de la popularidad ofrecerá resultados muy buenos, incluso difíciles de batir por algoritmos mucho más elaborados [Cremonesi et al., 2010]. Sin embargo, a veces esto lleva a situaciones subóptimas o injustas y se han publicado varios estudios para tratar de compensar estos sesgos. Por ejemplo, [Mehrotra et al., 2018] hace un esfuerzo para ofrecer igualdad de oportunidades a los vendedores en una plataforma de *e-commerce* independientemente de su popularidad. Por otro lado, [Cañamares and Castells, 2018] y [Mena-Maldonado et al., 2020] realizan un estudio probabilístico en el que se identifican situaciones en las que los sesgos de popularidad dan lugar a resultados no deseables, peores incluso que los que se conseguirían con recomendaciones aleatorias, y también a discrepancias entre *Precision* y *AntiPrecision*.

En este trabajo se distinguirá entre dos tipos de conjuntos de datos:

- **Datos no sesgados:** En este tipo de datos, poco comunes, los valores desconocidos se distribuyen de forma homogénea entre todos los ítems. Por eso, también son conocidos como conjuntos **Missing At Random (MAR)**. Esta situación es característica del conjunto *CM100k* [Cañamares and Castells, 2018]. Su distribución se muestra en la Figura 2.4(a). En ella, todos los ítems han recibido entre 70 y 130 ratings.
- **Datos sesgados:** En estos datos, mucho más frecuentes en la literatura, los *ratings* conocidos se concentran sobre una minoría de los ítems. Por tanto, hay muchos otros con muy pocas valoraciones. *MovieLens100k* es un conjunto de este tipo y su distribución de popularidad se muestra en la Figura 2.4(b). En este caso hay unos pocos ítems con más de 100 valoraciones y muchos con cantidades próximas a 0. Este tipo de datos también se denominan **Missing Not At Random (MNAR)**.

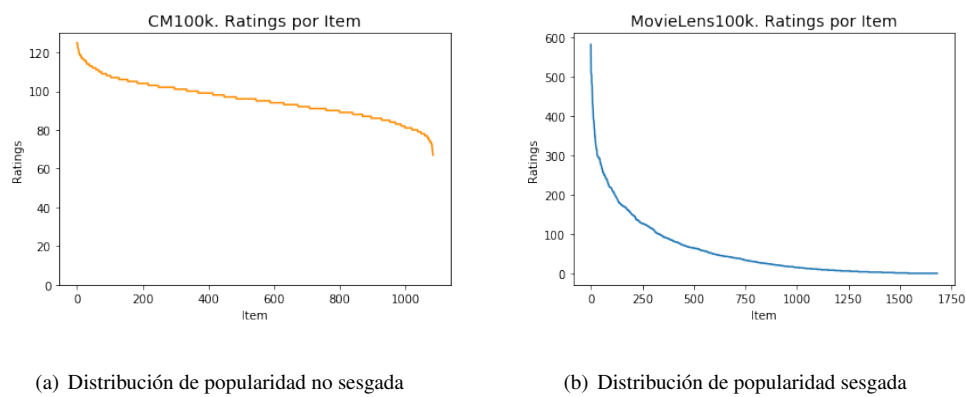


Figura 2.4: Sesgo de popularidad en la distribución de los *ratings* por ítem

2.2. Bandidos multibrazo

El problema del bandido multibrazo (MAB) o bandido de k brazos es un área de estudio muy conocida en el mundo de la probabilidad que destaca por la necesidad de plantear estrategias que balanceen beneficios inmediatos y potenciales a la hora de tomar decisiones. Su sencilla adaptación al paradigma del aprendizaje por refuerzo ha convertido a los *bandits* en una herramienta útil e intuitiva para afrontar problemas de decisión y, por ello, poco a poco han ganado presencia en la literatura con publicaciones como [Lattimore and Szepesvári, 2020; Slivkins, 2019; Sutton and Barto, 2018].

La primera referencia del problema del bandido [Thompson, 1933] - no así sus soluciones más conocidas - data de 1933 [Slivkins, 2019], cuando William R. Thompson, un entomólogo canadiense que estudiaba posibles aplicaciones de las matemáticas a la biología, se cuestionó cómo elegir el tratamiento que debe seguir un paciente cuando hay dos opciones disponibles y, aunque prometedora, una de ellas no está tan consolidada como la otra. Con el paso de los años, el mismo dilema va apareciendo en otras situaciones como los juegos de azar, y así surge la formulación más extendida

del problema:

Un cliente entra en un salón de juego que contiene k máquinas tragaperras dispuestas en hilera. El jugador, que tiene un presupuesto limitado, se pregunta en qué orden y cuántas veces debe jugar en cada una de ellas para maximizar sus ganancias.

Por supuesto, existen muchas variantes del problema: la probabilidad de éxito puede ir cambiando en cada máquina, alguna de ellas puede estar ocupada por otro jugador, en el casino deciden cambiar las máquinas más viejas por otras nuevas... pero, en general, no es posible conocer la solución óptima.

La aplicación de *bandits* a sistemas de recomendación es relativamente reciente, pero ha supuesto un tema de interés recurrente en la última década dada su habilidad innata para afrontar el problema de explotación *versus* exploración (Sección 2.1.3) [Cañamares et al., 2019; Dragone et al., 2019; Kawale et al., 2015; Li et al., 2010; Li et al., 2016; McInerney et al., 2018; Sanz-Cruzado et al., 2019; Wang et al., 2019; Zhao et al., 2013].

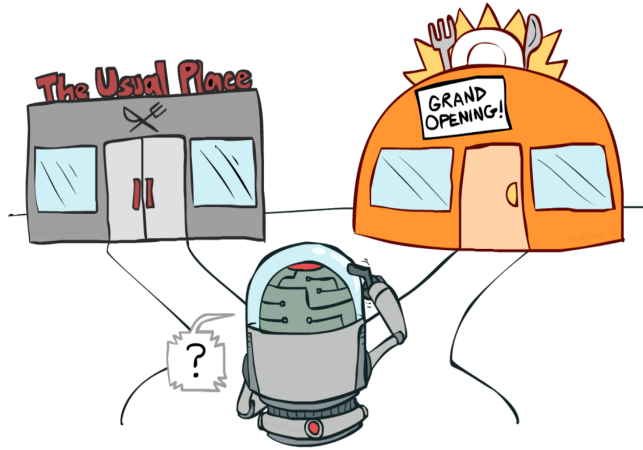


Figura 2.5: El dilema del bandido. Fuente: UC Berkeley AI course slide, lecture 11.

2.2.1. Formalización del problema del bandido

Supongamos que el jugador tiene k opciones. Estas opciones se llamarán brazos (*arms*) y la acción de ganar jugando en cada uno de ellos se representa por las variables aleatorias $A_1, A_2, A_3, \dots, A_k$. Sea T el número de partidas que se han jugado. Entonces:

- F_i es la función de distribución de probabilidad asociada a la v.a. A_i para todo $1 \leq i \leq k$. $\mu_i = \mathbb{E}[A_i]$ es la esperanza de A_i para todo $1 \leq i \leq k$, es decir, la esperanza de ganar tirando del brazo i . Las μ_i se conocen como el **valor** de sus respectivos brazos.
- α_i^t, β_i^t son, respectivamente, el número de éxitos y de fracasos obtenidos tirando del brazo i durante las t primeras acciones. Por tanto, $n_i^t = \alpha_i^t + \beta_i^t$ coincide con el número total de veces que se ha tirado del brazo A_i hasta el instante $1 \leq t \leq T$.

Conocidas F_i , $1 \leq i \leq k$, la forma razonable de actuar es maximizar la ganancia esperada, o lo que es lo mismo, elegir siempre el brazo con mayor μ_i . Sin embargo, lo normal es que las distribuciones de los brazos sean desconocidas y, por tanto, a la hora de tomar una decisión, será necesario estimar cada una de las esperanzas ($\hat{\mu}_i$) y evaluar la confianza que se tiene en dichas estimaciones. El proceso para calcular las estimaciones y cómo se tienen en cuenta las confianzas a la hora de elegir un brazo es lo que caracteriza a cada una de las estrategias para afrontar el problema del bandido, conocidas como *policies*. Está claro que, si solo se va a jugar una vez, la opción que maximiza la ganancia esperada es elegir el brazo con mayor $\hat{\mu}_i$, que es lo que se conoce como **estrategia avara (greedy)** o como explotación. No obstante, cuando se juega más de una vez, puede ser beneficioso explorar varias acciones hasta tener cierta confianza en las estimaciones y a partir de ese momento tomar decisiones avaras.

2.2.2. Estimaciones

La forma más común de estimar la recompensa esperada de cada brazo en cada instante t es:

$$\hat{\mu}_i^t = \frac{\alpha_i^t}{\alpha_i^t + \beta_i^t} \quad 1 \leq i \leq k \quad (2.11)$$

Para realizar una estimación en tiempo $t = 0$, es necesario definir manualmente los valores α^0 y β^0 . Esta inicialización suele ser la misma para todos los brazos y puede impactar de forma sustancial en los resultados que obtenga el *bandit*, así que es conveniente probar con varias combinaciones. Si se selecciona $\alpha^0 \gg \beta^0$ se estaría ante una **inicialización optimista** (más aciertos que fallos) y si, por el contrario, se elige $\alpha^0 \ll \beta^0$, entonces la **inicialización** sería **pesimista** (más fallos que aciertos) [Sutton and Barto, 2018].

2.2.3. Estrategias populares

A continuación se muestran tres de las estrategias de selección, también llamadas **políticas**, más conocidas para el problema del bandido. Todas ellas realizan un balance entre exploración y explotación, pero de forma muy distinta [Sutton and Barto, 2018].

ϵ -greedy

La estrategia ϵ -greedy es muy simple. Se elige la estrategia avara con probabilidad ϵ (explotación) y se elige una acción aleatoria con probabilidad $1 - \epsilon$ (exploración). Así, conforme crece el número de acciones, incluso los peores brazos se habrán seleccionado aproximadamente $\frac{N \cdot (1 - \epsilon)}{k}$ veces (Ecuación 2.12). Es normal configurar el parámetro ϵ para que decrezca con el tiempo. Un análisis de la eficacia y el comportamiento de esta estrategia en tiempo finito se puede encontrar en [Auer et al., 2002].

$$arm_t = \begin{cases} \text{random_integer}(1, k) & \text{si } \text{random}(0, 1) < \epsilon, \\ \underset{1 \leq i \leq k}{\operatorname{argmax}} (\hat{\mu}_i^t) & \text{en otro caso} \end{cases} \quad (2.12)$$

Upper Confidence Bound (UCB)

Como su propio nombre indica, *Upper Confidence Bound* realiza una estimación del valor máximo que podría llegar a tener un brazo teniendo en cuenta los resultados recogidos con anterioridad. Para seleccionar un brazo se utiliza la fórmula 2.13, basada en las desigualdades de Hoeffding y de Markov y propuesta en [Agrawal, 1995].

$$arm_t = \underset{1 \leq i \leq k}{\operatorname{argmax}} \left(\hat{\mu}_i^t + \sqrt{\delta \cdot \frac{\ln(t)}{n_i(t)}} \right) \quad (2.13)$$

La propia fórmula consta de dos términos. El primero de ellos coincide con la estimación de la esperanza estimada, por lo que se trata de un factor de explotación que da más peso a los brazos con mejores resultados. El segundo es un término de exploración por varios motivos: decrece cuanto más se ha explorado un brazo y, al mismo tiempo, crece de forma moderada con el número de iteraciones realizadas. Así, si el bandido se encuentra en un estado avanzado y uno de sus brazos se ha seleccionado pocas veces, este factor podría superar al valor más grande de entre todos los factores de exploración y beneficiar a un brazo poco conocido. El grado de exploración deseado se puede configurar por medio del parámetro δ , cuyo valor también puede decrecer con el tiempo. Para que la fórmula 2.13 tenga sentido, es necesario realizar una primera ronda de inicialización en la que todos los brazos se seleccionan una vez. [Auer et al., 2002] también estudia de forma analítica esta estrategia.

Thompson Sampling (TS)

La estrategia Thompson Sampling está basada en el concepto de distribución *Beta*, una familia de distribuciones de dos parámetros positivos α y β que contiene, entre otras, a la distribución uniforme en $[0, 1]$ ($\alpha = 1, \beta = 1$), a distribuciones muy parecidas a la normal cuando los dos parámetros tienen valores altos y similares entre sí o a otras que se comportan como una exponencial cuando hay grandes diferencias entre los dos parámetros (Figura 2.6).

En cada instante de tiempo t , Thompson Sampling modela los brazos del bandido de acuerdo a una distribución *Beta* tal que $A_i \sim \text{Beta}(\alpha_i^t, \beta_i^t)$, toma una muestra de cada una de las distribuciones y finalmente elige el brazo cuya muestra sea mayor (Ecuación 2.14). Las distribuciones *Beta* tienden a concentrarse cuanto mayor es $\alpha + \beta$ y se vuelven muy dispersas si esa suma es pequeña, por tanto, la propia distribución modela el concepto de confianza. Así, un brazo muy poco explorado (e.g. $\alpha = 1, \beta = 1$) puede tomar valores muy altos que beneficien su exploración. Además, como $\mathbb{E}[\text{Beta}(\alpha, \beta)] =$

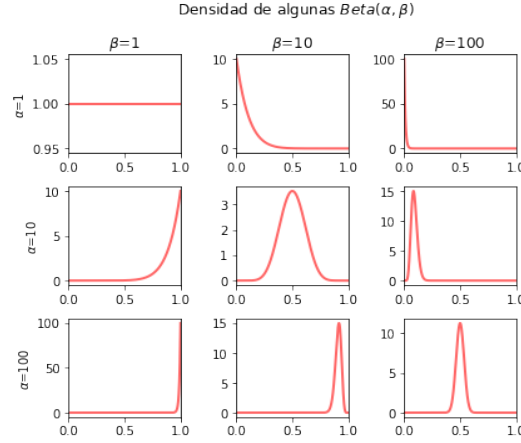


Figura 2.6: Funciones de densidad de algunas distribuciones Beta

$\frac{\alpha}{\alpha+\beta} = \hat{\mu}$, si hay un conocimiento amplio sobre A_i , los valores que tome la v.a. serán muy similares a la estimación natural del valor promedio. Una de las ventajas de TS es que, más allá de la inicialización α^0 y β^0 , no utiliza parámetros adicionales. Un análisis detallado de esta estrategia se puede encontrar en [Chapelle and Li, 2011].

$$arm_t = \operatorname{argmax}_{1 \leq i \leq k} (x \leftarrow \text{Beta}(\alpha_i^t, \beta_i^t)) \quad (2.14)$$

2.2.4. Adaptación a sistemas de recomendación

Los MAB se han adaptado a la tarea de recomendación de diversas formas. Por ejemplo, [López, 2019] propone un algoritmo no personalizado llamado *Item Bandit* y [Sanz-Cruzado et al., 2019] describe el algoritmo personalizado *kNN Bandit*. Ambos se detallan a continuación y serán empleados más tarde.

Item-Bandit

Item-Bandit es la adaptación natural de los MAB a la tarea de la recomendación: cada brazo representa a un ítem y se selecciona entre ellos empleando alguna de las políticas anteriormente mencionadas (ϵ -greedy, UCB o TS). Se trata de un algoritmo de recomendación no personalizada porque los valores de α y β asociados a cada brazo/ítem son comunes a todos los usuarios. [López, 2019] estudia de este algoritmo aplicando distintos protocolos y configuraciones, resultando en mejoras significativas sobre los algoritmos de referencia (Sección 2.1.5) en situaciones de arranque en frío. En el Algoritmo A.1 del Apéndice A se muestra el pseudocódigo de esta aproximación, en la línea 18 se llama a la función *policy*, que sería la aplicación directa de una de las Ecuaciones 2.12, 2.13 y 2.14.

kNN-Bandit

Esta versión combina un enfoque basado en *nearest neighbors* con un *bandit* que se utiliza para elegir los vecindarios. La idea principal de este algoritmo es la sustitución del cálculo tradicional de distancias de un método KNN (e.g. similitud coseno) por el uso de *Thompson Sampling* y el muestreo de una distribución *Beta*, lo que aporta un componente estocástico que facilita la exploración sin perder el grado de personalización. Este novedoso enfoque, propuesto en [Sanz-Cruzado et al., 2019] y cuyo código original se puede encontrar en [Sanz-Cruzado, 2019], obtiene, también en condiciones de arranque en frío, mejores resultados que los algoritmos *Item-Bandit* e incluso que los *kNN* y los de factorización de matrices estándar.

Es más sencillo comprender el caso base, con $k = 1$. Sean:

- $\alpha_{uv} = \alpha_{vu}$ el número de ítems valorados de igual forma por los usuarios u y v .
- β_{uv} el número de ítems valorados por v que no han recibido el mismo *feedback*, o directamente no tienen valoración, por parte de u . Nótese que $\beta_{uv} \neq \beta_{vu}$.
- n_v el número total de ítems valorados por v . En ese caso, $n_v = \alpha_{uv} + \beta_{uv}$.

Entonces, si el objetivo es recomendar a un usuario $u \in \mathcal{U}$, la elección de un ítem consta de dos pasos: la elección del recomendador (el vecino más próximo) y la recomendación del mismo:

$$nn(u) = \underset{\substack{v \in \mathcal{U} \\ v \neq u}}{\operatorname{argmax}} (x \leftarrow \operatorname{Beta}(\alpha_{uv}, n_v - \alpha_{uv})) \quad (2.15)$$

$$item(u) = \underset{i \in \mathcal{I}}{\operatorname{argmax}} (r(i, nn(u))) \quad (2.16)$$

La generalización a $k > 1$ se realiza de la siguiente manera: se eligen los k vecinos con valores más altos de forma análoga a la Ecuación 2.15 ($nn_1(u), \dots, nn_k(u)$) y se almacenan los resultados del muestreo (x_1, \dots, x_k). Para elegir el ítem, se realiza una media ponderada de las opiniones de dichos usuarios:

$$item(u) = \underset{i \in \mathcal{I}}{\operatorname{argmax}} \left(\sum_{j=1}^k x_j \cdot r(i, nn_j(u)) \right) \quad (2.17)$$

Su pseudocódigo, para un solo vecino, se muestra en el Algoritmo A.3 del Apéndice A. Se omite el cuerpo principal del proceso de recomendación por ser el mismo que en el Algoritmo A.1.

TRABAJO PREVIO

Antes de comenzar el proceso de experimentación ha sido necesario realizar una serie de tareas preliminares. En primer lugar, he desarrollado una herramienta adecuada para simular el comportamiento cíclico de los sistemas de recomendación capaz de recopilar las métricas descritas de la Sección 2.1.6. También he llevado a cabo un análisis de los conjuntos de datos que se van a emplear con el fin de describir sus rasgos más característicos. Además, se propone una ligera modificación en el algoritmo *Item Bandit*.

3.1. Una plataforma de recomendación: CycloRec

Pese a que existía la posibilidad de usar la plataforma de recomendación del grupo de *Information Retrieval* de la UAM, implementada en *Java*, se ha optado por realizar una implementación en *Python3*. Esta decisión se toma por dos motivos: porque una implementación propia otorga una comprensión mayor del funcionamiento de los algoritmos y porque, durante el proceso de búsqueda, no se encontró ninguna otra librería en *Python3* diseñada especialmente para soportar procesos de recomendación cíclicos. Es cierto que existe la librería *Surprise* [Hug, 2019], buena para recomendaciones en un paso y con una interfaz cómoda y extensible, pero no se adecua bien a un proceso iterativo. Por eso he implementado *CycloRec* [Cuesta, 2020], diseñada para procesos cíclicos y *feedback* explícito.

Funcionamiento general

Aunque una descripción más detallada del funcionamiento de *CycloRec* se puede encontrar en su propio repositorio, la Figura 3.1 muestra los bloques principales de su arquitectura.

Esencialmente, un sistema de recomendación en *CycloRec* tiene tres bloques: el primero, *Data-Layer*, almacena la lógica relacionada con la búsqueda y el almacenamiento de *ratings*; el segundo, o *UserLayer*, almacena la información relativa a cada usuario y el tercero, *Recommender*, abstrae el proceso de recomendar y de evaluar las recomendaciones realizadas. Las métricas disponibles son *Precision*, *Recall*, *Anti-Precision* y *Fallout*.

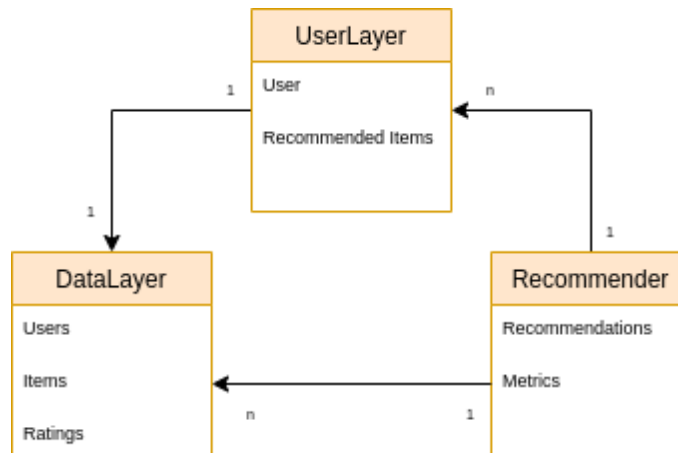


Figura 3.1: Bloques de CycloRec

En el Código 3.1 se muestra un ejemplo de uso en el que se realizan, empleando el algoritmo Item-Bandit ϵ -greedy, 1000 recomendaciones por usuario después de haber utilizado el 20 % del conjunto de *ratings* para entrenar, recopilando todas las métricas incluidas en *CycloRec* y evitando repeticiones.

Código 3.1: Ejemplo de uso de *CycloRec*

```

1  import pandas as pd
2  from cyclorec.data_layer.data_layer import DataLayer
3  from cyclorec.recommenders.simple.random_rec import ItemMABRecommender
4
5  # Reading data
6  ratings = pd.read_csv('example_ratings.csv')
7  # Creation of the data_layer
8  data_layer = DataLayer(name='example1', splitted=False, whole_set=ratings, test_proportion=0.2)
9  # Creation of the recommender
10 recommender = ItemMABRecommender(data_layer, policy='egreedy')
11 # Training
12 recommender.train()
13 # Recommendation loop
14 recommendations, metrics = recommender.recommendation_round_loop(maxiter=1000,
    enable_repetitions=False, enable_metrics=True)
  
```

Algoritmos implementados

Los algoritmos incluidos hasta el momento se resumen en la Tabla 3.1, no obstante, aquellos marcados con '**' tienen un coste computacional alto y es necesario trabajar más en su eficiencia. En total, se han implementado los tres algoritmos de referencia mencionados en la Sección 2.1.5, cuatro filtros colaborativos, incluyendo *kNN Bandit*, el algoritmo *Item Bandit* y una leve modificación del mismo (*IM Item Bandit*) que se describe a continuación.

Algoritmo	Abreviatura	Personalizado	Tipo	Políticas
Random	rand	\times	<i>Aleatorio</i>	-
Most Popular	pop	\times	<i>Greedy</i>	-
Average	avg	\times	<i>Greedy</i>	-
Item Bandit	item-bandit	\times	<i>Bandit</i>	ϵ -greedy, UCB, TS
IM Item Bandit	im-item-bandit	\times	<i>Bandit</i>	ϵ -greedy, UCB, TS
Item Based kNN*	item-knn	✓	<i>CF. Neighborhood</i>	-
User Based kNN*	user-knn	✓	<i>CF. Neighborhood</i>	-
HKV*	hkv	✓	<i>CF. Matrix factorization</i>	-
kNN Bandit*	knn-bandit	✓	<i>CF. Neighborhood. Bandit.</i>	TS

Tabla 3.1: Algoritmos incluidos en la librería *CycloRec*

Adaptación de Item-Bandit

Si uno observa el Algoritmo A.1 (Apéndice A), puede darse cuenta de que se plantea un dilema a la hora de actualizar los valores de α_i y n_i en las líneas 22 y 23. El problema surge cuando el ítem recomendado no tiene una valoración conocida por parte del usuario en \mathcal{R}_{test} . Si se opta por considerar esa situación como una valoración de 0, igual que en [López, 2019], entonces, por cada recomendación de i con *feedback* desconocido en \mathcal{R}_{test} se está incrementado de forma implícita β_i y, por lo tanto, aumenta, quizá de forma injusta, la creencia de que un ítem es malo. Por ejemplo, la ausencia de valoraciones puede deberse a que el ítem no es muy popular o es relativamente nuevo y no a que sea una mala opción. De igual manera, considerarlo como un valor positivo tendría un efecto perjudicial para los ítems populares, que perderían la ventaja que les otorga tener más recomendaciones. Por eso, se estudiará también una nueva versión que ignora los *ratings* que no están en el conjunto de test (*Ignore Missing Item-Bandit*), tal y como se indica en el Algoritmo A.2 (Apéndice A).

3.2. Análisis de los conjuntos de datos

3.2.1. MovieLens100k

	MovieLens100k	CM100k
Número de usuarios	943	1054
Número de ítems	1682	1084
Número de <i>ratings</i>	100.000	103.548
Número de <i>ratings</i> familiares	-	11.594
Valores de <i>ratings</i>	1,2,3,4,5	1,2,3,4

Tabla 3.2: Descripción de MovieLens100k y CM100k

Los conjuntos de datos *MovieLens* [Harper and Konstan, 2016] son una referencia en el campo de los sistemas de recomendación. Se basan en valoraciones que usuarios de la plataforma homónima han dado a diferentes películas. Existen varias opciones con distintas características, pero en este proyecto se emplea la versión *MovieLens100k*, cuyos rasgos principales se detallan en la Tabla 3.2.

Como es común, especialmente en los desarrollos *bandit* para recomendación, binarizamos los datos. De esta forma, los *ratings* con valores 1,2 o 3 se consideran negativos (0) y los *ratings* de 4 o 5 se consideran positivos (1). La distribución de los datos por sus valores se puede observar en la Figura 3.2(a) en las dos versiones.

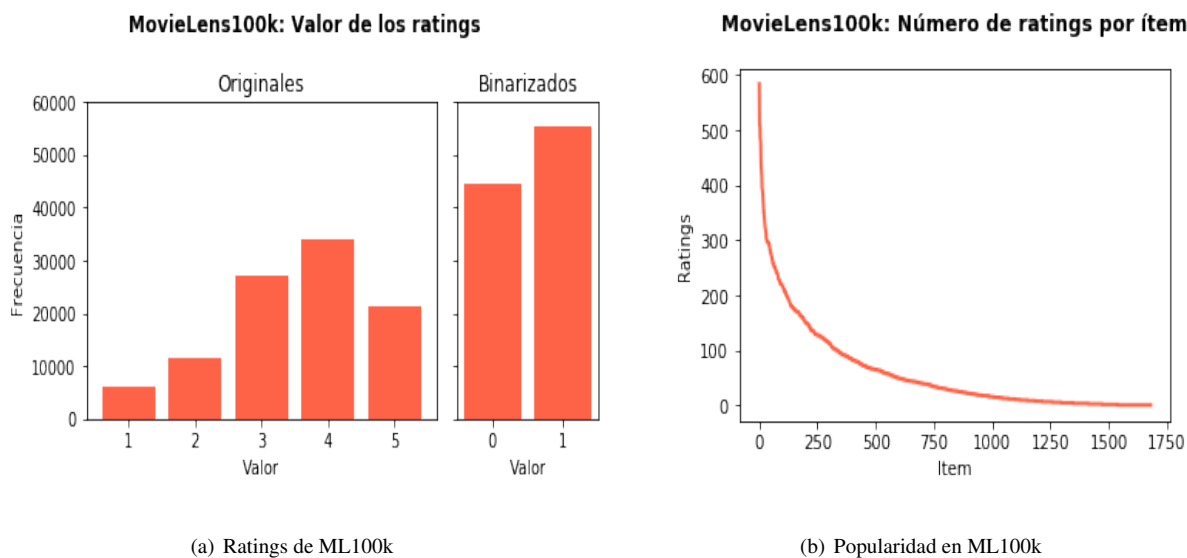


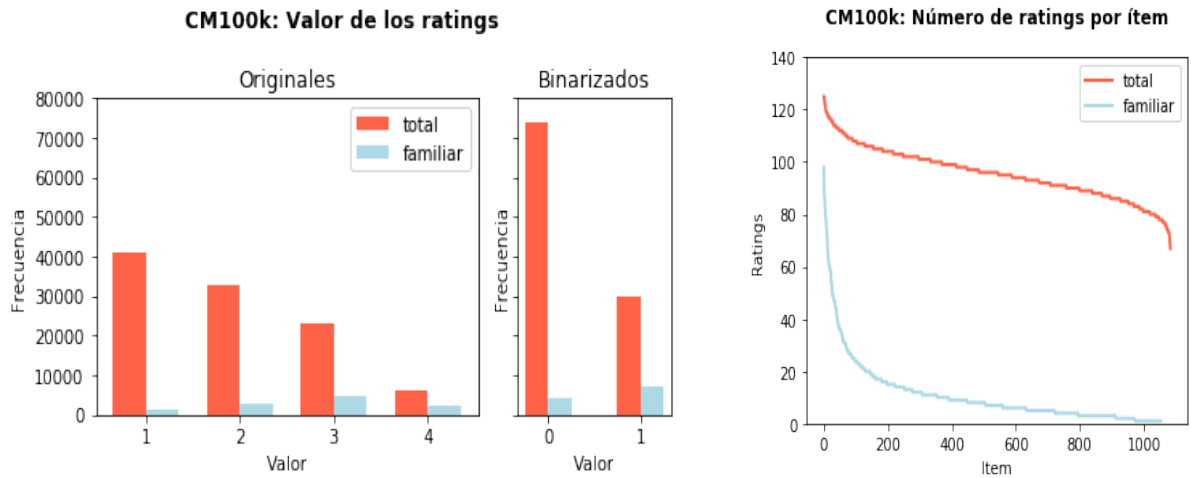
Figura 3.2: Distribución de los *ratings* en MovieLens100k por valores y por ítems

Por otro lado, es posible ver que los datos se distribuyen de forma MNAR (Sección 2.1.7) en la Figura 3.2(b), y, por tanto, se deduce la existencia de un sesgo de popularidad. Por ejemplo, hay pocos ítems con más de 400 valoraciones y muchos por debajo de las 50. Cuando se realicen experimentos con ML100k este será un factor a tener en cuenta.

3.2.2. CM100k

CM100k [Cañamares and Castells, 2018] es un conjunto de datos de recomendación en el ámbito musical elaborado mediante *crowdsourcing*. En este proceso se pidió a mil usuarios que valorasen cien ítems elegidos al azar de entre mil, de forma que los *ratings* se muestrearon con una distribución uniforme sobre los ítems. La peculiaridad de CM100k es que es lo suficientemente versátil como para reproducir tanto conjuntos sesgados como no sesgados. Esto es posible gracias a que también se preguntó a los encuestados si ya conocían cada una de las canciones. Se denomina *rating* familiar a cualquier *rating* en el que el usuario había escuchado la canción con anterioridad. En la Tabla 3.2 se muestran algunos valores característicos del conjunto.

Al igual que en el caso de *MovieLens100k*, se ha procedido a binarizar cada uno de los *ratings* considerando valoraciones positivas aquellas que valen 3 o 4 y negativas las demás. Las frecuencias para cada uno de los valores se muestran en la Figura 3.3(a).



(a) Ratings de CM100k

(b) Popularidad en CM100k

Figura 3.3: Distribución de los *ratings* en CM100k por valores y por ítems

En cuanto al sesgo de popularidad, en la Figura 3.3(b) se puede apreciar cómo los *ratings* familiares siguen una distribución sesgada o MNAR. Sin embargo, considerando la totalidad de los datos, se obtiene una distribución MAR (Sección 2.1.7) en la que cada ítem tiene en torno a 100 valoraciones, lo que no es muy común en RS y otorga interés al estudio de CM100k. El hecho de que la curva naranja no sea totalmente horizontal refleja la varianza de una distribución binomial, tomando una muestra por ítem, tantas repeticiones como usuarios y una probabilidad de éxito (presencia de *rating*) de en torno al 10 %.

EXPERIMENTACIÓN

Esta sección contiene tanto la descripción como los resultados de los experimentos realizados en el proyecto. Todos ellos tienen el fin de observar cómo se comportan los algoritmos basados en *bandits* y determinar qué tipo relaciones hay entre *Recall* y *Fallout* en procesos de recomendación cíclicos, siendo esto último algo que no se ha estudiado en ninguna de las referencias consultadas.

4.1. Metodología general

- Todas las simulaciones se realizan empleando *CycloRec*.
- Ninguno de los experimentos propuestos consta de fase de entrenamiento, con el objetivo de reproducir las condiciones de un arranque en frío.
- Todas las cifras ofrecidas son el promedio de los resultados obtenidos en tres repeticiones independientes.
- Durante el proceso de recomendación:
 - 1.– El número de rondas de recomendación (ciclo en el cual cada usuario recibe una única recomendación) considerado varía en función del experimento.
 - 2.– El orden de usuarios en cada ronda se determina de forma aleatoria.
 - 3.– No se recomienda el mismo ítem a un mismo usuario dos veces.
 - 4.– Para todos los modelos considerados, el *feedback* asociado a cada recomendación se consulta después de cada una de ellas y no al final de cada ronda.
- Durante el proceso de evaluación se calculan los valores de *Recall* y *Fallout* acumulados en cada una de las rondas de recomendación. Los resultados expuestos son el promedio, época a época, de todas las repeticiones computadas.

4.2. Experimento 1

Objetivo y diseño

Cualquier métrica puede emplearse para ordenar sistemas de recomendación de mejor a peor de acuerdo a los resultados de una evaluación *offline*. Es razonable pensar que una métrica y su anti-métrica, al medir fenómenos a priori complementarios, deberían ordenar distintos sistemas de forma equivalente: que el sistema que más aciertos consiga sea el que menos fallos cometa. Sin embargo, en [Mena-Maldonado et al., 2020] se concluye que, para recomendaciones en un paso y para la mayoría de *datasets* de recomendación, el *ranking* de sistemas proporcionado por la métrica *Precision* discrepa absolutamente del que surge al considerar *Anti-Precision*. Este experimento busca verificar si el mismo fenómeno ocurre también en procesos de recomendación iterativos, empleando otra pareja de métricas más adecuada para el paradigma cíclico como son *Recall* y *Fallout*. Para ello:

- Utilizamos el conjunto de datos MNAR *MovieLens100k*, que tiene el sesgo de popularidad característico de la mayoría de datos de recomendación.
- Los sistemas elegidos se basan en los algoritmos **aleatorio**, **popularidad**, **promedio** (Sección 2.1.5), **kNN Bandit**, **ϵ -greedy Item Bandit**, **UCB Item Bandit**, **TS Item Bandit** (Sección 2.2.4), **ϵ -greedy Item Bandit (IM)**, **UCB Item Bandit (IM)** y **TS Item Bandit (IM)** (Sección 3.1).
- Para obtener buenas configuraciones de cada uno de los algoritmos *bandit*, se realizará una búsqueda en rejilla sobre los parámetros involucrados en cada uno de ellos, tomando los valores indicados en la Tabla 4.1. Se toma $k = 1$ porque suele ofrecer buenos resultados [Sanz-Cruzado et al., 2019]. Para configurar el resto de parámetros se utilizarán los valores en $t = 500$, que es un tiempo lo suficientemente lejano como para haberse sobrepuesto al arranque en frío y que permite valorar también los sistemas a medio plazo. No se trata de un proceso de entrenamiento, porque las pruebas posteriores comenzarán, de nuevo, sin conocimiento previo. Como el objetivo de este experimento es estudiar el comportamiento de las métricas y no el de los algoritmos, no es tan relevante obtener la configuración óptima para cada uno de ellos.

Parámetro	Valores
α_0	0, 1, 10, 100, 1000
β_0	0, 1, 10, 100, 1000
ϵ	0.1, 0.2, 0.4, 0.8
δ	0.01, 0.1, 1, 10
k	1

Tabla 4.1: Rejilla para cada parámetro

- A partir de los resultados de la búsqueda en rejilla, se seleccionan para cada algoritmo dos

configuraciones, la que maximiza el *Recall* acumulado y la que minimiza el *Fallout*. Además, para cada estrategia (*policy*) de los *bandits*, se toma el método de actualización que mejores resultados tiene, es decir, se elige solamente uno entre **item-bandit-policyX** e **im-item-bandit-policyX**.

- Después, se ejecutan los algoritmos seleccionados para el ciclo de recomendación completo desde el principio, 1682 épocas en este caso, para comprender mejor su evolución.
- Para finalizar, se elaboran dos *rankings* de sistemas según los valores de *Recall* y *Fallout* obtenidos en $t = 500$, cuya discrepancia será medida con el coeficiente de Kendall (τ).

Búsqueda en rejilla

Los resultados de la búsqueda en rejilla se presentan en las Tablas 4.2 y 4.3, conteniendo la primera las configuraciones que maximizan *Recall* y la segunda las que minimizan *Fallout*. En cada una de ellas se remarca el método de actualización seleccionado y los valores que motivan dicha elección.

Algoritmo	Policy	α_0	β_0	$\epsilon/\delta/k$	Recall	Fallout
Item Bandit	ϵ -greedy	1	10	0.1	0.809	0.609
Item Bandit (IM)	ϵ -greedy	1	1000	0.2	0.785	0.581
Item Bandit	UCB	0	0	0.01	0.807	0.606
Item Bandit (IM)	UCB	1	100	0.01	0.799	0.604
Item Bandit	TS	10	1000	-	0.803	0.604
Item Bandit (IM)	TS	10	1000	-	0.798	0.599
kNN Bandit	TS	10	100	1	0.862	0.677

Tabla 4.2: Configuraciones que maximizan Recall en $t = 500$ para MovieLens100k

Algoritmo	Policy	α_0	β_0	ϵ/δ	Recall	Fallout
Item Bandit	ϵ -greedy	1000	1	0.4	0.350	0.319
Item Bandit (IM)	ϵ -greedy	1000	1	0.1	0.086	0.092
Item Bandit	UCB	0	0	100	0.328	0.310
Item Bandit (IM)	UCB	10	1	0.01	0.159	0.103
Item Bandit	TS	1000	1	-	0.346	0.319
Item Bandit (IM)	TS	1000	1	-	0.087	0.097
kNN Bandit	TS	100	1	1	0.385	0.366

Tabla 4.3: Configuraciones que minimizan Fallout en $t = 500$ para MovieLens100k

Sobre los resultados anteriores, llama la atención que un mayor *Recall* implique un mayor *Fallout*. El objetivo era maximizar *Recall* y minimizar *Fallout*, pero en este caso parece que las buenas recomendaciones van de la mano de malas recomendaciones. Es curioso también que sean las configuraciones *Ignore Missing* las que minimizan el *Fallout*, es decir, las que se equivocan menos. Al no penalizarse las acciones sin recompensa asociada en el conjunto de test, ítems que tienen pocos pero buenos *ratings* se recomiendan con mucha más frecuencia. Como consecuencia, se realizan muchas más recomendaciones sin feedback y por ello tanto la tasa de acierto como la de fallo son tan bajas.

Resultados en ciclo completo

La Figura 4.1 muestra el ciclo completo de los algoritmos con la configuración optimizada para *Recall* de la Tabla 4.2, además de incluir las referencias **Popularidad** y **Promedio**. Por cuestiones de visualización, a su *Recall* acumulado se le ha restado el *Recall* acumulado correspondiente al sistema **Aleatorio**. Si bien todos los *bandits* superan sin problema al algoritmo de **Popularidad**, es cierto que, a excepción de **kNN Bandit**, no se mejoran los resultados de **Promedio** de forma sustancial, como ya anticipó [Cremonesi et al., 2010].

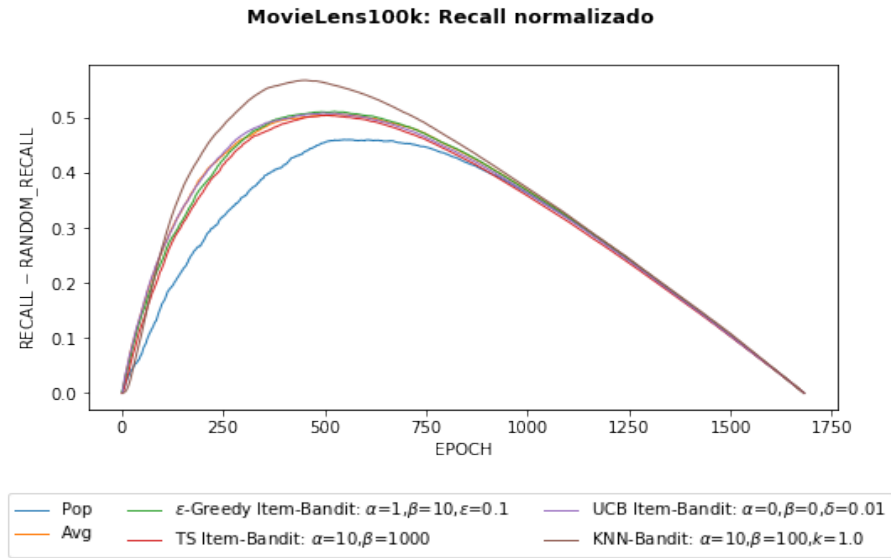


Figura 4.1: Recall normalizado en ML100k de algoritmos optimizados para Recall

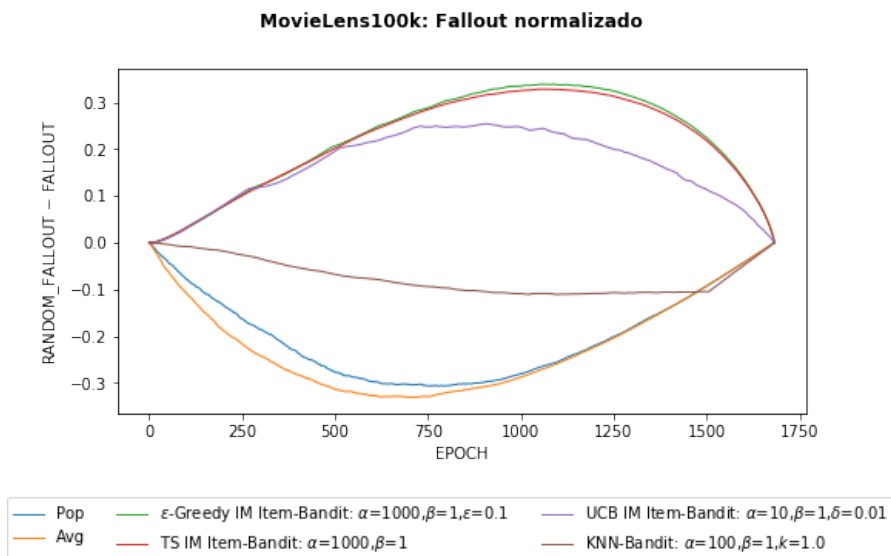


Figura 4.2: Fallout normalizado en ML100k de algoritmos optimizados para Fallout

Por su parte, la Figura 4.1 ofrece el *Fallout* aleatorio menos el *Fallout* de los algoritmos optimizados para minimizarlo de la Tabla 4.3. Los algoritmos con valores negativos se equivocan más que el algoritmo aleatorio, y es el caso de **Popularidad y Promedio**. Esto remarca la importancia de tener en cuenta las métricas de falsos positivos ya que, dependiendo del entorno, puede resultar crucial evitar esos índices de fallos. Por otro lado, las versiones *Item Bandit* para minimizar *Fallout* se equivocan entre un 20 % y un 30 % menos en el momento de mayor diferencia. **kNN Bandit**, que destacaba en términos de *Recall*, se equivoca ahora más que la recomendación aleatoria, siendo entonces una mala opción para la tarea de minimizar fallos.

Recall versus Fallout

En la Figura 4.3 se muestran los sistemas ordenados de acuerdo a los valores que toman *Recall* y *Fallout* en $t = 500$ acompañados de una visualización que une cada sistema del *ranking* de *Recall* con la posición que ocupa el mismo sistema en el *ranking* de *Fallout*.

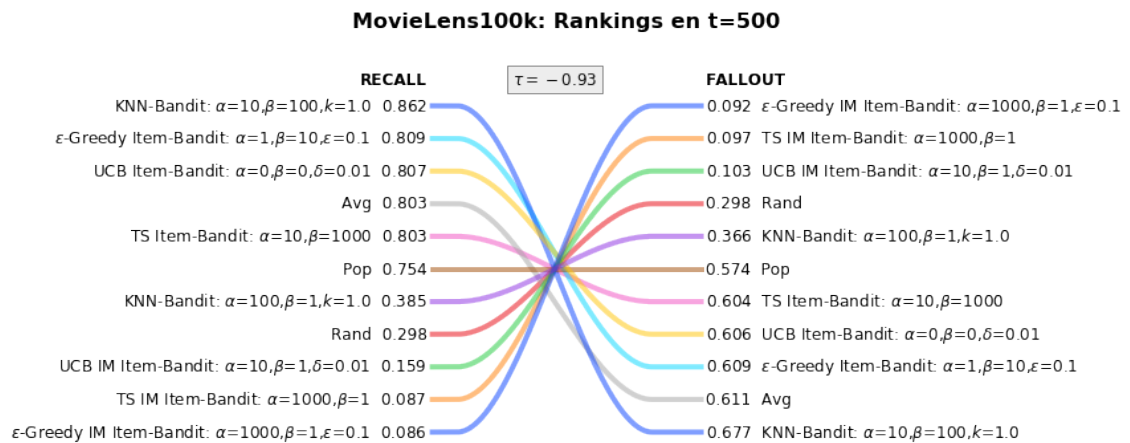


Figura 4.3: Rankings de algoritmos en ML100k

A tenor de los resultados, se pueden extraer varias conclusiones. La primera es que, fijado un objetivo, existen *bandits* que mejoran sustancialmente el algoritmo de popularidad y, aunque por menos, el de valoración promedio. La segunda es que la actualización *Ignore Missing* es mucho más eficaz a la hora de minimizar el *Fallout*, aunque obtiene resultados muy pobres en términos de *Recall*. Y la tercera es que, al menos en *MovieLens100k*, *Recall* y *Fallout* no solo no están de acuerdo si no que ofrecen *rankings* radicalmente opuestos, alcanzándose una correlación de *Kendall* de $\tau = -0.93$. Por esto último, parece sensato pensar que lo observado en [Mena-Maldonado et al., 2020] es válido también para procesos cíclicos y otras parejas métrica-antimétrica.

4.3. Experimento 2

El objetivo de este experimento es determinar si la discrepancia observada en el Experimento 1 se reproduce sobre otros conjuntos de datos, tanto con el sesgo de popularidad como sin él.

Para ello, se utilizará ahora el dataset *CM100k* en sus dos variantes. Seleccionando solo los datos “familiares” (Sección 3.2), es posible conseguir un dataset MNAR, como el de *MovieLens100k*, que denominamos **Familiar CM100k**. Por el contrario, si se seleccionan todos los *ratings*, se obtiene un poco frecuente conjunto MAR. Este experimento consta de dos pruebas diferenciadas: primero se repetirá el procedimiento del Experimento 1 sobre el subconjunto MNAR y luego se hará lo propio con el conjunto MAR. Los algoritmos considerados son los mismos que en el Experimento 1 y la rejilla de búsqueda también coincide (Tabla 4.1) pero en este caso el ciclo completo constará de 1084 iteraciones, tantas como ítems en el sistema.

4.3.1. Familiar CM100k

Búsqueda en rejilla

Las configuraciones que maximizan *Recall* se presentan en la Tabla 4.4. En este caso, no existe un método de actualización que se imponga sustancialmente al otro porque para la política UCB la versión *IM* obtiene mejores resultados. También se aprecia que los valores altos de *Recall* vuelven a ir acompañados de valores altos de *Fallout*, es decir, tienen altas tasas de fallo, al igual que en el Experimento 1.

Algoritmo	Policy	α_0	β_0	ϵ/δ	Recall	Fallout
Item Bandit	ϵ -greedy	1	1	0.1	0.720	0.665
Item Bandit (IM)	ϵ -greedy	1	100	0.4	0.702	0.643
Item Bandit	UCB	0	0	0.01	0.690	0.632
Item Bandit (IM)	UCB	1	100	0.01	0.712	0.649
Item Bandit	TS	1	1000	-	0.708	0.653
Item Bandit (IM)	TS	10	1000	-	0.707	0.657
kNN Bandit	TS	1	1000	1	0.678	0.621

Tabla 4.4: Configuraciones que maximizan Recall en $t = 500$ para Familiar CM100k

Por otro lado, las configuraciones que minimizan *Fallout* (Tabla 4.5) son todas *IM*, concordando también con lo observado en el Experimento 1. Además, para estas últimas configuraciones, no se obtienen valores *Fallout* y *Recall* tan bajos como los de la Tabla 4.3, aunque esto puede deberse a que el número total de *ratings* por descubrir es mucho menor.

Algoritmo	Policy	α_0	β_0	ϵ/δ	Recall	Fallout
Item Bandit	ϵ -greedy	1000	100	0.2	0.471	0.464
Item Bandit (IM)	ϵ -greedy	1000	1	0.1	0.322	0.297
Item Bandit	UCB	1000	100	1	0.466	0.453
Item Bandit (IM)	UCB	10	1	0.01	0.345	0.307
Item Bandit	TS	1000	1	-	0.464	0.463
Item Bandit (IM)	TS	100	1	-	0.314	0.303
kNN Bandit	TS	10	1	1	0.485	0.471

Tabla 4.5: Configuraciones que minimizan Fallout en $t = 500$ para Familiar CM100k

Resultados en ciclo completo

El ciclo completo de los algoritmos de la Tabla 4.4, optimizados para *Recall*, se muestra en la Figura 4.4. Los resultados vuelven a ser muy similares a los observados para MovieLens100k, con **Promedio** difícil de batir. En este caso, sin embargo, *kNNBandit* ofrece malos resultados, no supera ni siquiera al algoritmo por popularidad.

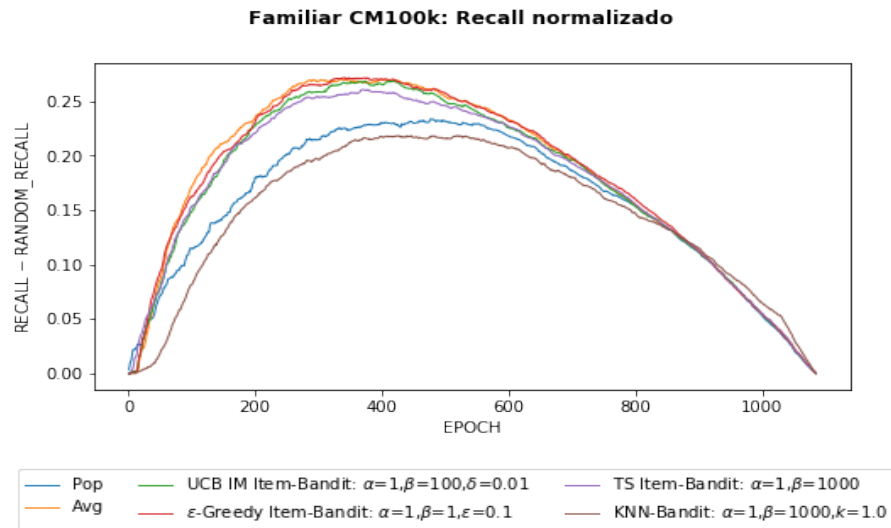


Figura 4.4: Recall normalizado en Familiar CM100k de los algoritmos optimizados para Recall

El comportamiento también es el mismo que en el Experimento 1 para el caso de *Fallout* (4.5). Las referencias **Popularidad** y **Promedio** tienen unos resultados malos, cometiendo muchos más fallos, en torno a un 15 % más, que una recomendación aleatoria en las 500 primeras épocas. Por su parte, como es lógico, los *bandits* diseñados para ello son mucho más eficaces evitando dar recomendaciones erróneas.

Recall versus Fallout

Por último, la Figura 4.6 muestra la ordenación de todos los algoritmos anteriores en función de su *Recall* y *Fallout*, respectivamente. Y de nuevo, como en el caso de *MovieLens100k* (Figura 4.3), los

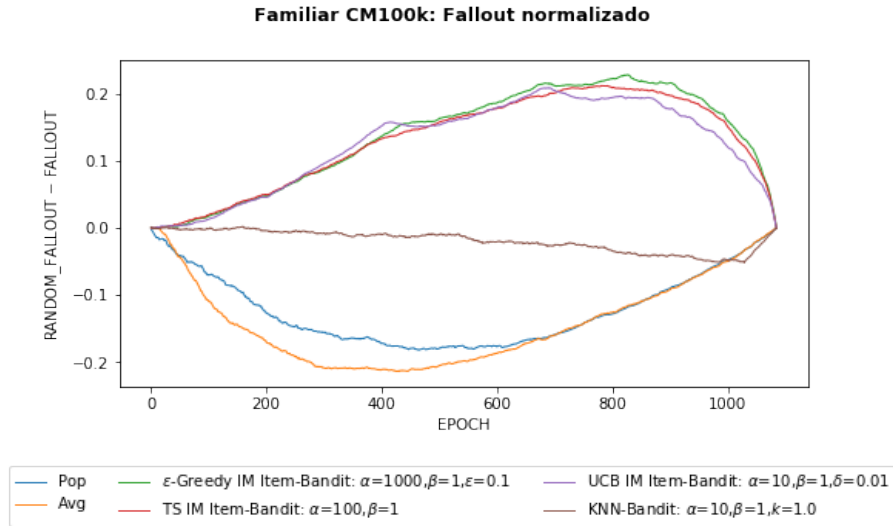


Figura 4.5: Fallout normalizado en Familiar CM100k de los algoritmos optimizados para Fallout

rankings resultantes son diametralmente opuestos ($\tau = -0.83$). Otra vez, en un conjunto MNAR, una métrica y su antimétrica ofrecen resultados contrarios. A falta de una aproximación analítica, lo propuesto por [Mena-Maldonado et al., 2020] parecer ser cierto también para recomendaciones iterativas.

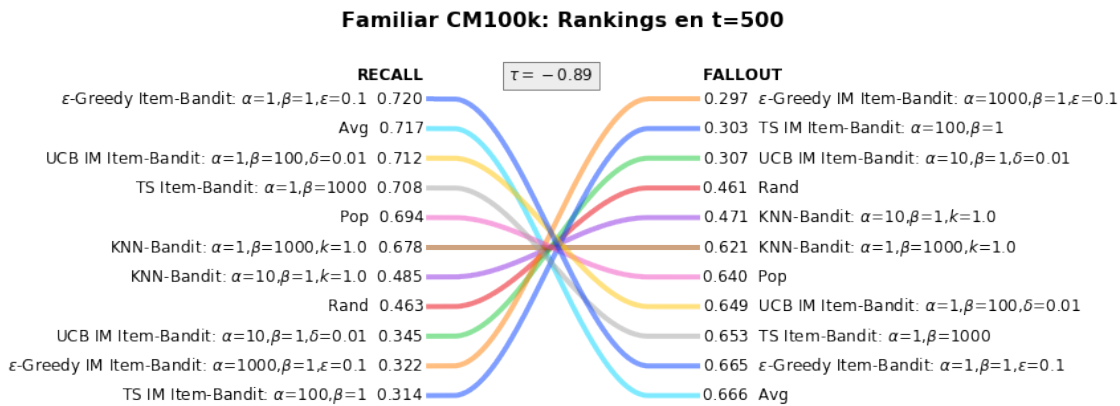


Figura 4.6: Rankings de algoritmos en Familiar CM100k

4.3.2. CM100k

Búsqueda en rejilla

El caso de CM100k ofrece unos resultados muy distintos a lo que se había observado en las pruebas anteriores: las configuraciones que maximizan *Recall* son ahora las mismas que minimizan *Fallout*. Los algoritmos seleccionados se muestran en la Tabla 4.6.

Si en el Experimento 1 y en la fase 1 del Experimento 2 parecía que las versiones de *Item Bandit* con update *Ignore Missing* eran mejores para minimizar *Fallout* pero tenían un mal desempeño a

Algoritmo	Policy	α_0	β_0	ϵ/δ	Recall	Fallout
Item Bandit	ϵ -greedy	1	1	0.2	0.577	0.414
Item Bandit (IM)	ϵ -greedy	1	1	0.1	0.587	0.410
Item Bandit	UCB	0	0	0.01	0.577	0.414
Item Bandit (IM)	UCB	1	1	0.1	0.587	0.410
Item Bandit	TS	10	1000	-	0.576	0.416
Item Bandit (IM)	TS	1	10	-	0.581	0.411
kNN Bandit	TS	1	1000	1	0.552	0.423

Tabla 4.6: Configuraciones que maximizan *Recall* y minimizan *Fallout* en $t = 500$ para CM100k

la hora de maximizar *Recall*, en este caso son las que mejor se comportan en ambas situaciones, superando a todos los algoritmos de referencia y a *kNNBandit*. Como CM100k es un conjunto MAR, no existen muchos ítems populares que concentren los *ratings*. Por tanto, la penalización por *rating* desconocido que se hace de forma implícita afecta de igual manera a todos los ítems e incrementa la diferencia entre la estimación y la esperanza real de acertar seleccionado cada ítem. Las versiones “IM” parecen ser capaces de solucionar este problema.

Resultados en ciclo completo

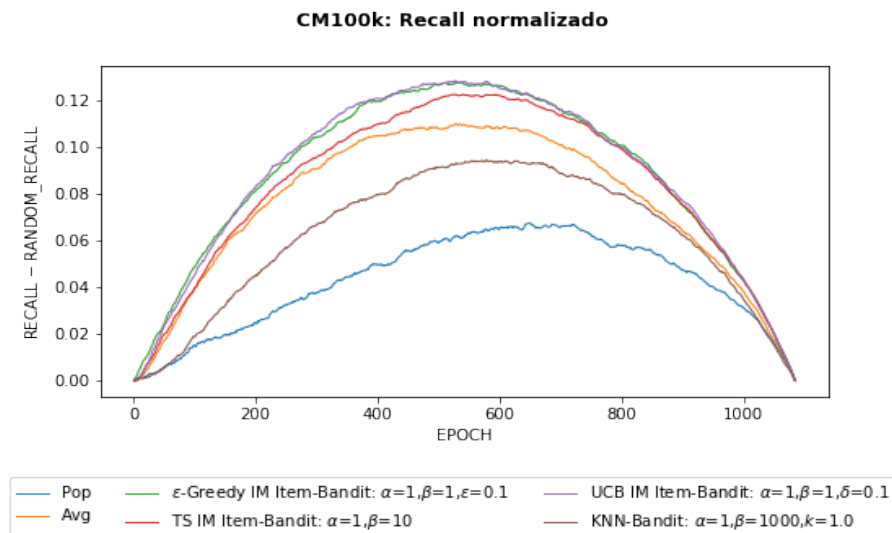


Figura 4.7: Recall normalizado en CM100k para algoritmos que optimizan Recall y Fallout

Las Figuras 4.7 y 4.8 presentan, respectivamente, *Recall* y *Fallout* obtenidos por los algoritmos de la Tabla 4.6, con la diferencia de que, en esta ocasión, en ambas gráficas aparecen las mismas configuraciones. En ellas destacan dos cuestiones:

- 1.— En este caso los *bandits* sí que mejoran de forma significativa el resultado del algoritmo **Promedio** prácticamente durante el ciclo completo. Además, tanto **Popularidad** como **Promedio** mejoran a la recomendación aleatoria en términos de *Fallout*, algo que no ocurría

evaluando con conjuntos MNAR.

2.– La dos gráficas comparten forma, en el sentido en que *Fallout* y *Recall* evolucionan de forma paralela para cada una de las configuraciones. Esto quiere decir que una mejora en el *Recall* con respecto a **Aleatoria** implica también una mejora, de acuerdo a cierta proporción, en *Fallout* con respecto a **Aleatoria**.

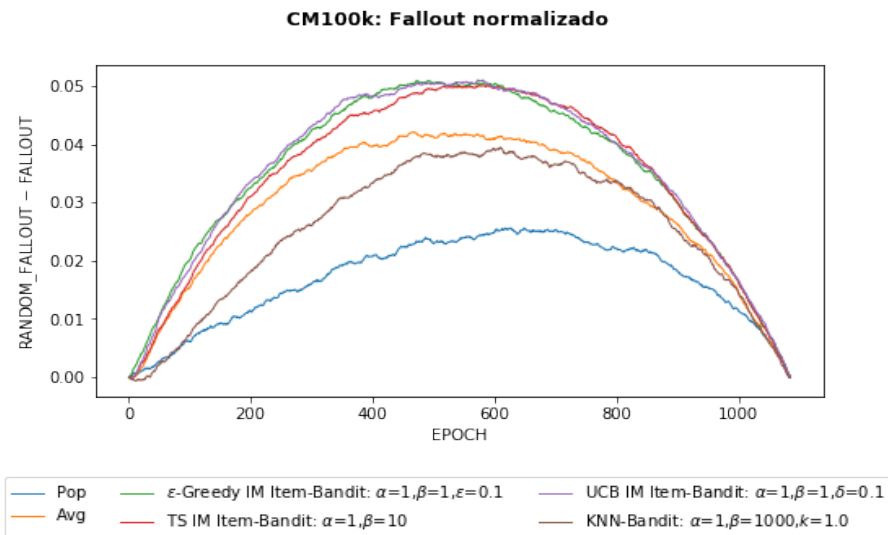


Figura 4.8: Fallout normalizado en CM100k para algoritmos que optimizan Recall y Fallout

Recall versus Fallout

Como se podía intuir a partir de lo observado en las secciones anteriores, el *ranking* resultante de maximizar *Recall* y el *ranking* resultante de minimizar *Fallout* tienden a estar de acuerdo (Figura 4.9). Es más, en este caso, la ordenación es exactamente la misma, lo que resulta en una $\tau = 1$. Por lo tanto, en conjuntos MAR, parece que maximizar el número de recomendaciones buenas sí que equivale a minimizar el número de recomendaciones malas en términos de métricas TP y FP.

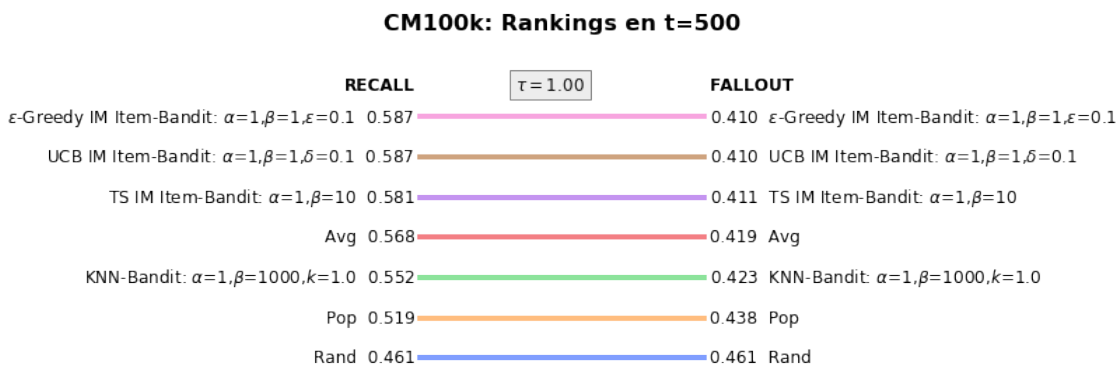


Figura 4.9: Rankings de algoritmos en CM100k

CONCLUSIONES Y LIMITACIONES

5.1. Conclusiones

El objetivo principal de este proyecto era determinar cuándo la evaluación de sistemas de recomendación cíclicos mediante métricas de verdaderos positivos concuerda con la que se obtiene a partir de métricas de falsos positivos. Para ello, se han utilizado los clásicos algoritmos de referencia y otros más novedosos basados en banditos multi-brazo, dos conjuntos de datos con distribuciones y sesgos de popularidad distintos, y las métricas *Recall* y *Fallout*.

En relación con este objetivo, todos los experimentos realizados indican que el grado de acuerdo o desacuerdo entre *Recall* y *Fallout* depende de la distribución de popularidad de los ítems en el conjunto de test. En conjuntos MNAR (MovieLens100k y Familiar CM100k), las evaluaciones realizadas por ambas métricas no solo no están de acuerdo sino que la discrepancia en la ordenación de sistemas es absoluta. Si se usa este tipo de datos es necesario, pues, prestar especial atención a los falsos positivos, ya que maximizar el número de recomendaciones buenas puede implicar obtener también un número peligroso de recomendaciones negativas. En conjuntos MAR (CM100k), *Recall* y *Fallout* sí se comportan de forma coherente entre sí y un mayor acierto se vincula a una tasa de fallo menor. Todo esto concuerda con lo expuesto en [Cañamares and Castells, 2018] y [Mena-Maldonado et al., 2020] para recomendaciones de un solo paso y las métricas *Precision* y *Anti-Precision*.

Respecto a la aplicación de *bandits* a sistemas de recomendación, en concreto mediante las distintas versiones de *Item Bandit*, se ha comprobado empíricamente que, si el objetivo es evitar recomendaciones desagradables, son mejores aproximaciones las que no penalizan un ítem por no tener *feedback* asociado en el conjunto de test. Además, en conjuntos MAR, esta apreciación se aplica también al objetivo de maximizar el número de buenas recomendaciones. La diferenciación de estos dos métodos de actualización de *bandits* no se llevaba a cabo en ningún trabajo de la literatura consultada y estudiar sus efectos con mayor detalle puede resultar útil para mejorar los resultados de los banditos en el futuro.

5.2. Limitaciones y trabajo futuro

A lo largo del desarrollo de este proyecto se han realizado una serie de simplificaciones sobre el problema de la recomendación que limitan la aplicación práctica de los resultados obtenidos. Por ello, sería conveniente realizar más experimentos con datos de carácter implícito, que son la base de una parte muy importante del mercado de los sistemas de recomendación a día de hoy, como es el caso de *Spotify* o *Netflix*.

También existen otros puntos que, bien por limitaciones de tiempo o por no estar alineados con el objetivo principal del proyecto, quedan pendientes. Entre ellos se encuentran:

- Ampliación del catálogo librería *CycloRec* y mejora del rendimiento de varios de los algoritmos implementados hasta el momento, en especial los KNN o HKV. Consideración de la posibilidad de crear *ensembles* o combinaciones entre todos estos algoritmos.
- Modificación de *CycloRec* para permitir que las recomendaciones se realicen de forma paralela, lo que aceleraría considerablemente la experimentación.
- Ampliación del catálogo de *datasets* con conjuntos más variados en términos de la distribución de popularidad.
- Consideración de otras condiciones más allá del arranque en frío, con distintas proporciones de entrenamiento.
- Desarrollo de un marco matemático que justifique lo observado.

Por otro lado, al haberse desarrollado una línea de investigación bastante concreta como es el grado de acuerdo/desacuerdo entre métricas TP y métricas FP, hay otros posibles enfoques cuya investigación sigue abierta, por ejemplo:

- Determinar los contextos en los que una mala recomendación es realmente crítica.
- Estudiar de forma profunda la utilidad y las limitaciones de los *bandits* en sistemas de recomendación.
- Profundizar en el concepto de sesgo e identificar cuando es positivo dejarse llevar por la popularidad.

BIBLIOGRAFÍA

- [Adomavicius and Tuzhilin, 2005] Adomavicius, G. and Tuzhilin, A. (2005). Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions.
- [Agrawal, 1995] Agrawal, R. (1995). Sample Mean Based Index Policies with $O(\log n)$ Regret for the Multi-Armed Bandit Problem. Technical Report 4.
- [Auer et al., 2002] Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256.
- [Baumeister et al., 2001] Baumeister, R. F., Bratslavsky, E., Finkenauer, C., and Vohs, K. D. (2001). Bad is stronger than good. *Review of General Psychology*, 5(4):323–370.
- [Bellogín et al., 2017] Bellogín, A., Castells, P., and Cantador, I. (2017). Statistical biases in information retrieval metrics for recommender systems. *Information Retrieval Journal*, 20(6):606–634.
- [Bron et al., 2019] Bron, M., Haines, A., Zhou, K., and Lalmas, M. (2019). Uncovering bias in ad feedback data analyses & applications. In *The Web Conference 2019 - Companion of the World Wide Web Conference, WWW 2019*, pages 614–623. Association for Computing Machinery, Inc.
- [Brost et al., 2019] Brost, B., Mehrotra, R., and Jehan, T. (2019). The music streaming sessions dataset. In *The Web Conference 2019 - Proceedings of the World Wide Web Conference, WWW 2019*, pages 2594–2600. Association for Computing Machinery, Inc.
- [Burke, 2007] Burke, R. (2007). Hybrid web recommender systems. In *Lecture Notes in Computer Science (including sub-series Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 4321 LNCS, pages 377–408. Springer Verlag.
- [Cañamares and Castells, 2018] Cañamares, R. and Castells, P. (2018). Should I follow the crowd? A probabilistic analysis of the effectiveness of popularity in recommender systems. In *SIGIR' 18: The 41st International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 415–424. Association for Computing Machinery, Inc.
- [Cañamares et al., 2020] Cañamares, R., Castells, P., and Moffat, A. (2020). Offline evaluation options for recommender systems. *Information Retrieval Journal*, pages 1–29.
- [Cañamares et al., 2019] Cañamares, R., Redondo, M., and Castells, P. (2019). Multi-armed recommender system bandit ensembles. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 432–436. Association for Computing Machinery, Inc.
- [Chapelle and Li, 2011] Chapelle, O. and Li, L. (2011). An Empirical Evaluation of Thompson Sampling. In *Proceedings of the 24th International Conference on Neural Information Processing Systems, NIPS'11*, pages 2249–2257, Red Hook, NY, USA. Curran Associates Inc.
- [Cremonesi et al., 2010] Cremonesi, P., Koren, Y., and Turrin, R. (2010). *Performance of Recommender Algorithms on Top-N Recommendation Tasks*. ACM.
- [Cuesta, 2020] Cuesta, E. (2020). CycloRec source code: <https://github.com/emiliocuestaf/CycloRec>.
- [Dragone et al., 2019] Dragone, P., Mehrotra, R., and Lalmas, M. (2019). Deriving user-and content-specific rewards for contextual bandits. In *Proceedings of the 2019 World Wide Web Conference (WWW' 19)*, pages 2680–2686. Association for Computing Machinery, Inc.
- [Fields, 2011] Fields, B. (2011). Contextualize Your Listening: The Playlist as Recommendation Engine. Technical report.
- [Harper and Konstan, 2016] Harper, F. M. and Konstan, J. A. (2016). The MovieLens Datasets. *ACM Transactions on Interactive Intelligent Systems*, 5(4).
- [Hu et al., 2008] Hu, Y., Koren, Y., and Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. In *Eighth IEEE International Conference on Data Mining*. Institute of Electrical and Electronics Engineers (IEEE), Institute of Electrical and Electronics Engineers.
- [Hug, 2019] Hug, N. (2019). Surprise library: <http://surpriselib.com/>.

- [Kawale et al., 2015] Kawale, J., Bui, H., Kveton, B., Thanh, L. T., and Chawla, S. (2015). Efficient Thompson Sampling for Online Matrix-Factorization Recommendation. Technical report.
- [Lattimore and Szepesvári, 2020] Lattimore, T. and Szepesvári, C. (2020). *Bandit Algorithms*. Cambridge University Press.
- [Li et al., 2010] Li, L., Chu, W., Langford, J., and Schapire, R. E. (2010). A contextual-bandit approach to personalized news article recommendation. *WWW '10: Proceedings of the 19th International Conference on World Wide Web*, pages 661–670.
- [Li et al., 2016] Li, S., Karatzoglou, A., and Gentile, C. (2016). Collaborative filtering bandits. *SIGIR 2016 - Proceedings of the 39th international ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 539–548.
- [López, 2019] López, E. (2019). Bandidos multi-brazo en sistemas de recomendación. Technical report, Escuela Politécnica Superior. Universidad Autónoma de Madrid.
- [McInerney et al., 2018] McInerney, J., Lackner, B., Hansen, S., Higley, K., Bouchard, H., Gruson, A., and Mehrotra, R. (2018). Explore, exploit, and explain: personalizing explainable recommendations with bandits. In *RecSys 2018: Proceedings of the 12th ACM Conference on Recommender Systems*, pages 31–39. Association for Computing Machinery, Inc.
- [Mehrotra et al., 2018] Mehrotra, R., McInerney, J., Bouchard, H., Lalmas, M., and Diaz, F. (2018). Towards a fair marketplace: counterfactual evaluation of the trade-off between relevance, fairness & satisfaction in recommendation systems. In *International Conference on Information and Knowledge Management (CIKM' 2018)*, pages 2243–2251. Association for Computing Machinery.
- [Mena-Maldonado et al., 2020] Mena-Maldonado, E., Castells, P., Ren, Y., Sanderson, M., and Cañamares, R. (2020). Agreement and disagreement between true and false-positive metrics in recommender systems evaluation. *Proceedings of the 43rd ACM International Conference on Research and Development in Information Retrieval (SIGIR 2020)*, pages 841–850.
- [Pilászy et al., 2010] Pilászy, I., Zibriczky, D., and Tikk, D. (2010). Fast ALS-based matrix factorization for explicit and implicit feedback datasets. In *Conference: Proceedings of the 4th ACM Conference on Recommender Systems*, pages 71–78.
- [Pizzato et al., 2013] Pizzato, L., Rej, T., Akehurst, J., Koprinska, I., Yacef, K., and Kay, J. (2013). Recommending people to people: The nature of reciprocal recommenders with a case study in online dating. *User Modelling and User-Adapted Interaction*, 23(5):447–488.
- [Ricci et al., 2011] Ricci, F., Rokach, L., Shapira, B., and Kantor, P. (2011). *Recommender systems handbook*. Springer US.
- [Sánchez and Bellogín, 2018] Sánchez, P. and Bellogín, A. (2018). Measuring anti-relevance: A study on when recommendation algorithms produce bad suggestions. In *RecSys 2018 - 12th ACM Conference on Recommender Systems*, pages 367–371. Association for Computing Machinery, Inc.
- [Sanz-Cruzado, 2019] Sanz-Cruzado, J. (2019). kNN Bandit source code: <https://github.com/ir-uam/kNNBandit>.
- [Sanz-Cruzado et al., 2019] Sanz-Cruzado, J., Castells, P., and López, E. (2019). A simple multi-armed nearest-neighbor bandit for interactive recommendation. In *RecSys 2019 - 13th ACM Conference on Recommender Systems*, pages 358–362. Association for Computing Machinery, Inc.
- [Slivkins, 2019] Slivkins, A. (2019). Introduction to Multi-Armed Bandits.
- [Sutton and Barto, 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning : an introduction*. The MIT Press, 2 edition.
- [Thompson, 1933] Thompson, W. R. (1933). On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of Two Samples. Technical Report 3.
- [Wang et al., 2019] Wang, Q., Zeng, C., Zhou, W., Li, T., Iyengar, S. S., Schwartz, L., and Grabarnik, G. Y. (2019). Online Interactive Collaborative Filtering Using Multi-Armed Bandit with Dependent Arms. *IEEE Transactions on Knowledge and Data Engineering*, 31(8).
- [Yin et al., 2010] Yin, D., Bond, S., and Zhang, H. (2010). Are bad reviews always stronger than good? Asymmetric negativity bias in the formation of online consumer trust. In *Thirty First International Conference on Information Systems, St. Louis 2010*. ICIS 2010 Proceedings.
- [Zhao et al., 2013] Zhao, X., Zhang, W., and Wang, J. (2013). Interactive collaborative filtering. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management - CIKM '13*, New York, New York, USA. ACM Press.

ACRÓNIMOS

FN False Negatives.

FP False Positives.

KNN *k nearest neighbors*.

MAB bandidos multi-brazo.

métricas FP métricas de falsos positivos.

métricas TP métricas de verdaderos positivos.

MNAR missing not at random.

RS sistemas de recomendación.

TN True Negatives.

TP True Positives.

TS *Thompson sampling*.

UCB *upper confidence bound*.

APÉNDICES

PSEUDOCÓDIGOS

A.1. Item Bandit

```

1  input :  $\mathcal{R}_{test}$ : Conjunto de ratings en test.  $\alpha^0 \in \mathbb{N}$ .  $\beta^0 \in \mathbb{N}$ .
2  output:  $metrics$ . Colección de métricas computadas.
3  begin
4     $\mathcal{T} \leftarrow \mathcal{U} \times \mathcal{I}$ 
5     $\mathcal{R}_{train} \leftarrow \emptyset$ 
6     $metrics \leftarrow 0$ 
7    foreach  $i \in \mathcal{I}$  do  $\alpha_i \leftarrow \alpha^0$ ;  $n_i \leftarrow \alpha^0 + \beta^0$ 
8     $t \leftarrow 0$ 
9    while  $t < N$  &  $\mathcal{T} \neq \emptyset$  do
10     foreach  $u \in \mathcal{U}$  do
11        $i \leftarrow \text{recomendar}(\mathcal{R}_{train}, u, \alpha, n, t)$ 
12        $\mathcal{T} \leftarrow \mathcal{T} \setminus \{(u, i)\}$ 
13        $\mathcal{R}_{train} \leftarrow \mathcal{R}_{train} \cup \{(u, i, \text{rating}(\mathcal{R}_{test}, u, i))\}$ 
14        $metrics \leftarrow \text{actualizar\_metricas}(\mathcal{R}_{test}, u, i)$ 
15        $\text{actualizar\_modelo}(\mathcal{R}_{test}, \mathcal{R}_{train}, u, i, \alpha, n)$ 
16     end
17      $t \leftarrow t + 1$ 
18   end
19 end
20 function  $\text{recomendar}(\mathcal{R}_{train}, u, \alpha, n, t)$  :
21    $arm \leftarrow \text{policy}(u, \alpha, n, t)$ 
22   return  $arm$ 
23 function  $\text{actualizar\_modelo}(\mathcal{R}_{test}, \mathcal{R}_{train}, u, i, \alpha, n)$  :
24    $\alpha_i \leftarrow \alpha_i + \text{rating}(\mathcal{R}_{test}, u, i)$ 
25    $n_i \leftarrow n_i + 1$ 
26   return

```

Algoritmo A.1: Item Bandit Recommender para condiciones de arranque en frío.

A.2. Ignore Missing Item Bandit

```
1 function actualizar_modelo(  $\mathcal{R}_{test}, \mathcal{R}_{train}, u, i, \alpha, \beta, n, \text{ } :$ 
2    $rat \leftarrow \text{rating}(\mathcal{R}_{test}, u, i)$ 
3   if  $rat$  is not null &  $rat == 1$  then
4      $\alpha_i \leftarrow \alpha_i + 1$ 
5   else if  $rat$  is not null &  $rat == 0$  then
6      $\beta_i \leftarrow \beta_i + 1$ 
7   end
8    $n_i \leftarrow n_i + 1$ 
9   return
```

Algoritmo A.2: Ignore Missing Item-Bandit update

A.3. kNN Bandit

```
1 function recomendar(  $\mathcal{R}_{train}, u, \alpha, n, t$  ) :
2   foreach  $v \in \mathcal{U} \setminus \{u\}$  do  $x_v \leftarrow \text{muestra\_beta}(\alpha_{vu}, n_v - \alpha_{vu})$ 
3    $v \leftarrow \underset{v \in \mathcal{U} \setminus \{u\}}{\text{argmax}}(x_v)$ 
4   return  $\underset{i \in \mathcal{I}}{\text{argmax}}(\text{rating}(\mathcal{R}_{train}, v, i))$ 
5 function actualizar_modelo(  $\mathcal{R}_{test}, \mathcal{R}_{train}, u, i, \alpha, n$  ) :
6    $n_u \leftarrow n_u + 1$ 
7   foreach  $v \in \mathcal{U} \setminus \{u\}$  do
8     if  $(v, i) \in \mathcal{R}_{train}$  then
9        $\alpha_{uv} \leftarrow \alpha_{uv} + \text{rating}(\mathcal{R}_{test}, u, i) \cdot \text{rating}(\mathcal{R}_{train}, v, i)$ 
10    end
11  end
12  return
```

Algoritmo A.3: kNN Bandit Recommender con $k = 1$ para condiciones de arranque en frío.